IP MULTICAST USING QUALITY OF SERVICE ENABLED OVERLAY NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Marc Brogle

von Sisseln AG

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

IP MULTICAST USING QUALITY OF SERVICE ENABLED OVERLAY NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Marc Brogle

von Sisseln AG

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 25.05.2010

Der Dekan:
Prof. Dr. Urs Feller

Abstract

In this thesis, we will present solutions for end users on how to profit from efficient data transmission and dissemination using multicasting with Quality of Service (QoS) support. Multicasting is one of the most efficient mechanisms to distribute data from one sender to multiple receivers concurrently. Since IP Multicasting is not widely deployed in the Internet today, Overlay Multicast has been introduced to overcome this limitation. It is often also referred to as Application Layer Multicast (ALM). ALM is though not a standard as IP Multicast is, and many different ALM protocols targeted for different application scenarios exist. Therefore, we developed a bridge between IP Multicast and Overlay Multicast, called the Multicast Middleware. Now, end users can benefit from applications supporting the IP Multicast API, while still be able to use them through the Internet. This bridging also maps IP Multicast traffic to unicast packet flows which makes it easier to support QoS for multicasting. To provide the best service to end users, QoS mechanisms for ALM protocols are also required. The OM-QoS (Quality of Service for Overlay Multicast) framework aims to enable QoS for different ALM protocols, facilitating the creation of QoS supporting multicast trees. We will present protocol dependent as well as protocol independent solutions to support QoS for different ALM protocols. Finally, we present MCFTP (Multicast File Transfer Protocol), which enables collaborating data dissemination to profit from the benefits of multicast distribution. It facilitates using resources available in the network (routers) or at end systems (for ALM) more efficiently.

Contents

List of Figures

List of Tables

Preface

The work presented in this thesis has been performed during my five-years employment as research and lecture assistant at the Institute of Computer Science and Applied Mathematics (IAM) of the University of Bern, Switzerland. The research conducted in this work has been partially supported by EuQoS Integrated Project of the European Union 6th Framework Programme under contract IST FP6 IP 004503.

I would like to thank first my advisor, Prof. Dr. Torsten Braun, head of the Computer Network and Distributed Systems group (RVS), who supervised my work. I would also like to thank Prof. Dr. Pascal Felber for accepting to read and judge this work, and both Prof. Dr. Oscar Nierstrasz and Prof. Dr. Edmundo Monteiro, who were willing to be co-examiners of this work.

Many thanks go to my working colleagues and friends at the institute and in our research group. It was a great joy to have worked with them in a friendly and pleasant working environment. In particular, I would like to thank Carlos Anastasiades, Markus Anwander, Thomas Bernoulli, Kirsten Dolfus, Marc Heissenbüttel, Philipp Hurni, Patrick Lauer, Dragan Milic, Matthias Scheidegger, Marc-Alain Steinemann, Thomas Staub, Markus Wälchli, Gerald Wagenknecht, Attila Weyland, Markus Wulff, and Zhongliang Zhao. I am also very grateful to Dragan Milic, with whom I shared the office room during the five years of my PhD, who also worked with me in the EuQoS project, and always was open for discussion about my ideas and approaches.

I also would like to thank all the students, who contributed to this thesis in one or more ways. In particular, thanks go to Sebastian Barthlomé, Luca Bettosini, Ulrich Bürgi, Alican Geycasar, Milan Nikolic, Dominic Papritz, and Andreas Rüttiman, who performed their bachelor's and/or master's thesis with me. Furthermore, I also like to thank Ruth Bestgen, the secretary of the RVS research group, for her help and support during all these years. Finally, many thanks go to Peppo Brambilla, the IAM's system administrator, for joining our research group during many lunches and for his patience and support on many system administration related issues.

I am very grateful to my family and my girlfriend Zora Lazarov, who supported me in many ways. Furthermore, I would like to thank my friend Raphael Reitmann, with whom I played squash during many years to compensate for the brain-centric research.

Chapter 1

Introduction

1.1 Overview

In this thesis, we are looking for an answer to the following question:

> "How can end users benefit from Internet-wide and QoS supporting multicast services to efficiently disseminate data?"

The answer to this question offers a solution to three basic but connected problems and is provided by our three main contributions:

- *Quality of Service for Overlay Multicast* to, e.g., guarantee bandwidth for best user Quality of Experience (QoE);
- *Providing IP Multicast Services via Peer-to-Peer (P2P) / Overlay Networks* to bridge IP Multicast with Application Layer Multicast;
- *Efficient Data Dissemination using Cooperation* of users to improve overall data dissemination performance.

Our *Quality of Service for Overlay Multicast* framework "OM-QoS" aims to enable QoS for different ALM protocols. It facilitates the creation of QoS supporting multicast trees. Therefore, the constructed P2P network and Application Layer Multicast (ALM) topology also support QoS requirements. We investigated several different Overlay Multicast protocols and came up with a series of protocol dependent as well as protocol independent solutions to support QoS for ALM.

For *Providing IP Multicast Services via P2P / Overlay Networks*, we developed a bridge between IP Multicast and Overlay Multicast called the "Multicast Middleware". Therefore, end users can use the widely available IP Multicast API that is supported by many applications. But, at the same time, end users are also able to benefit from the use of multicasting mechanisms throughout the Internet. This mapping of IP Multicast traffic to simple unicast packet flows makes it also easier to support Quality of Service (QoS) for multicasting. Since the unicast connections between the group members are established using a P2P network and are

known in advance, unicast QoS reservations can be performed easily if the network supports those.

To support *Efficient Data Dissemination using Cooperation*, we developed Multicast File Transfer Protocol "MCFTP". It enables more efficient usage of resources (of routers and/or end systems), in order to distribute files among interested users. This is achieved by using multicasting mechanisms to improve overall system performance and to reduce bandwidth consumption as well as download time.

1.2 Problem Statement

To distribute data from one sender to multiple receivers efficiently and concurrently, multicasting is one of the most appropriate mechanisms. Unfortunately, IP Multicasting, which would facilitate efficient group based communication in the Internet, is not widely deployed. Thus, it can not be used by end users in the Internet today. This is due to multiple reasons [61, 154, 63], which include:

- complex billing agreements between Internet Service Providers (ISP)
- security concerns (Denial of Service attacks, traffic volume control, etc.)
- configuration complexity (IP Multicast intra- and inter-domain routing setup)
- performance considerations (simple and high performance routers vs. complex routers to support IP Multicast)

Application Layer Multicast (ALM), often also referred to as Overlay Multicast, has been introduced to overcome the limitations of IP Multicast availability. ALM use Peer-to-Peer (P2P) networks among the group members to distribute the multicast data.

To give the best service to end users, Quality of Service (QoS) mechanisms for P2P/ALM networks are desirable to enhance user experience for applications like:

- Internet TV (IPTV)
- real-time multimedia (audio/video) broadcasting and streaming
- Massively Multiplayer Online Games (MMOG)
- Networked Virtual Environments (NVE)
- online collaboration tools

Various approaches [107, 105, 185, 106, 3, 189, 101] have been proposed to support a certain degree of QoS in P2P/ALM networks. Unfortunately, these approaches are often limited to improve (hop-by-hop) latencies [3], offering redundant paths, or selecting local group clusters trying to support bandwidth requirements of users [185]. Also, these different P2P/ALM protocols with limited QoS support are

normally only limited to specific application scenarios. Therefore, they are only working for, e.g., low bandwidth video streaming or delay restricted small-group collaboration networks. Generally, these solutions lack general QoS support for multiple QoS aspects in different application scenarios. They are often limited to provide the best possible service to end users rather than trying to match QoS requests with appropriate QoS guarantees.

Data dissemination with collaborating users has also become very popular. BitTorrent is one of the most used protocols used for file exchange between users. It though relies only on unicast connections between the collaborating users. But, such efficient and not concurrent file distribution mechanisms through the Internet can also benefit from the multicasting paradigm to improve overall performance.

1.3 Quality of Service for Overlay Multicast

To give the best service to end users, Quality of Service (QoS) mechanisms for P2P/ALM networks would be desirable. This would enhance user experience for multimedia applications using P2P/ALM, such as multimedia audio/video (A/V) streaming, IPTV services, massively multiplayer online games, real-time A/V conferencing, collaboration tools, networked virtual environments, etc.

We offer a solution to this problem by providing the Quality of Service for Overlay Multicast framework called "OM-QoS". OM-QoS enables different P2P / ALM protocols to support QoS. It basically enables the different protocols to build multicast trees that are QoS aware and are built with a QoS supporting structure.

In order to support QoS for many different P2P/ALM protocols, we analyzed various protocols, such as Scribe/Pastry, NICE, CAN, Chord, ZigZag, Tapestry. We designed protocol dependent solutions for Scribe/Pastry, NICE, and Chord, but also found a protocol independent solution that can be applied to arbitrary P2P networks. The protocol independent solution has been evaluated using CAN and NICE, but would work with any other P2P/ALM protocols as well.

To support QoS for P2P/ALM networks, we introduce the concept of QoS classes, which facilitates aggregation of multiple QoS parameters into one discrete QoS class value. Some restrictions though apply to this concept: it can only support parameters that are independent of the path length. This means that parameters that depend on multiple hops, such as the round trip times (RTT), cannot be supported by our presented QoS classes. Therefore, we considered RTT requirements separate from QoS classes. The node's RTT requirements together with the QoS class concept allow us to cover many QoS scenarios, where we can guarantee bandwidth as well as end-to-end delays.

The results of the evaluations show that OM-QoS indeed enables QoS support for the presented P2P/ALM protocols. Using OM-QoS, we were able to guarantee QoS parameters, such as bandwidth or node to root RTT to the individual nodes participating in the overlay network. This resulted in constructing multicast trees that support these QoS requirements.

1.4 Providing IP Multicast Services via P2P / Overlay Networks

ALM using P2P overlay networks could solve the problem of sparse IP Multicast support in the Internet. A limitation of this approach is the lack of standardized interfaces for existing IP Multicast applications. We propose a solution that bridges ALM and IP Multicast and uses a P2P overlay network to transport multicast data.

Our solution uses an ALM for transporting multicast traffic over the Internet. At the same time it offers a standard IP Multicast interface to applications on end systems. This solution is based on the so-called Multicast Middleware. The Multicast Middleware enables the transparent use of ALM mechanisms for all IP Multicast enabled applications on end systems. This is achieved by using a virtual network interface intercepting and forwarding multicast packets to the Multicast Middleware. For the end system the virtual network interface seems to be a normal Ethernet device attached to a fully featured Ethernet network with an IP Multicast router. But in reality our Multicast Middleware provides these services through the virtual network interface to the end system.

This mechanism can be used for high bandwidth and real time multimedia streaming using, e.g., the IP Multicast enabled application Video LAN Client (VLC). Generally, this approach allows end users to benefit from IP Multicast applications and efficient multicast data dissemination without requiring additional service deployment in the Internet to support IP Multicast.

To transmit the data using Overlay Multicast, we used Freepastry, a freely available implementation of the Scribe/Pastry ALM/P2P protocols. We implemented the major part of the Multicast Middleware in a platform independent way (using JAVA) supporting various operating system platforms, such as Windows, Linux and Mac OS X. The virtual network interface was provided by using TUN/TAP, which required only to implement a small operating specific component to communicate with the Multicast Middleware.

We evaluated the Multicast Middleware regarding many aspects, such as throughput, latency, loss, etc. Our evaluations show that the Multicast Middleware is a viable solution for end users to support high-bandwidth and real-time supporting IP Multicast applications. This includes audio/video streaming, IPTV services, collaboration tools, massively multiplayer online games, networked virtual environments, etc.

Mapping multicast communication to only unicast connections facilitates supporting QoS more easily. The Multicast Middleware can perform QoS reservations in the underlaying network if these are provided by QoS systems.The mechanisms presented for *Quality of Service for Overlay Multicast* using the OM-QoS framework for Scribe/Pastry were integrated to support constructing QoS-enabled multicast trees.

1.5 Efficient Data Dissemination using Cooperation

There are many different approaches (e.g., Bullet [100], FastReplica [53], Slurpie [157], BitTorrent [26]) on how data among multiple downloaders / subscribers can be distributed in an efficient way. The goal is to reduce download time of the individual downloaders even if the server or provider is overloaded, or if no such central provider is available at all.

Normally, these approaches rely on the cooperation of the downloaders by redistributing their already partially downloaded data. Therefore, users interested in specific data join a general or dedicated network. Data is then exchanged among the peers, either with the help of a resource provider outside the P2P network or completely isolated inside the P2P network.

A very popular protocol to distribute data efficiently is BitTorrent. Users share their resources to improve overall download performance. But, BitTorrent and many other solutions use only unicast connections. Therefore, they can not benefit from the efficiency provided by the multicast paradigm.

To improve resource usage in the network or at the end systems, we developed MCFTP, a Multicast File Transfer Protocol. Like BitTorrent, files are broken down into pieces and distributed among peers in the P2P network. But, in contrast to BitTorrent, MCFTP uses multicast communication to distribute the pieces. It supports either IP Multicast or ALM communication to disseminate the data.

We compared the performance of MCFTP to BitTorrent using the ns2 network simulator in IP Multicast and Overlay Multicast scenarios. The focus of these evaluations was on small to large scale networks. Furthermore, we also implemented a prototype of MCFTP using JAVA. The prototype implementation based on Freepastry was evaluated in a local testbed. It supports IP Multicast as well as Overlay Multicast in the form of Scribe/Pastry. We compared our prototype implementation of MCFTP to Azureus, which is a popular BitTorrent client available for many operating system platforms. The focus of the prototype evaluation was on small scale networks and to validate a proof of concept implementation of the MCFTP protocol for the Internet.

The results of the evaluations using the network simulator and using the prototype implementation in a local testbed show that MCFTP is a viable alternative to BitTorrent for efficient data dissemination using multicasting and collaboration. MCFTP performs better regarding download times and end system resource usage in terms of bandwidth. Hence, using multicast for data dissemination not only improves download times but also reduces overall resource usage on routers and/or end systems by decreasing bandwidth consumption.

1.6 Contributions

We built a framework that enables QoS support for various P2P/ALM protocols such as Scribe/Pastry, CAN, NICE, Chord, etc. using protocol dependent and protocol independent solutions. While the protocol dependent approach exploits specific P2P/ALM protocol features to introduce QoS, the protocol independent approach can be applied to almost any P2P/ALM in order to enable QoS. Various QoS parameters, such as bandwidth requirements and RTT constraints are supported. A particular feature of our solution is the support of QoS classes, which facilitates combining various QoS parameters into one distinct value. Hence, end users can now benefit from QoS support for various applications, such as multimedia streaming, massively multiplayer online games, networked virtual environments, conferencing, etc. We showed that a majority of the end-to-end paths in a P2P/ALM network using our framework support the QoS requirements of the individual end systems. Our framework only introduces a slight overhead in terms of multicast hop count, fan-out, node to root RTT, etc.

Furthermore, we built middleware bridging IP Multicast and P2P / ALM. Hence, end users can benefit from IP Multicast through the Internet, even though their Internet service providers or the networks along the transmission path might not support it. A key characteristic of our middleware is that it runs on various operating systems (Mac OS X, Win32, Linux) and supports high bandwidth applications with transmission rates above 100 Mbps. With our solution, we can support different high-bandwidth and delay sensitive scenarios, such as real-time video broadcasting, IPTV services, massive multiplayer online games, and others. Also, the Multicast Middleware has been combined with our QoS framework. This enables existing IP Multicast applications to work Internet-wide as well as to support QoS. Hence, we can now build multicast trees using P2P/ALM to support QoS requirements and capabilities of end systems while offering the widely used and standardized IP Multicast API to applications.

Additionally, we designed a multicast based protocol for data dissemination using cooperation, which allows users to exchange data more efficiently. This protocol can be used with IP Multicast or P2P/ALM. It has been designed to support various P2P/ALM protocols. Overall download times are reduced and also the bandwidth usage in end systems and in routers is decreased compared to traditional data dissemination protocols. One peculiarity of this protocol is that it can run in centralized and decentralized modes, which have their individual advantages. A high degree of optimization can be achieved in the centralized version, where one entity has the overview of all entities involved in the data dissemination. The decentralized version is more scalable and more robust regarding node failures, since the strategy and algorithms for dissemination are distributed. Finally, using the previous contributions presented, our protocol can be enhanced to support QoS in order to further enhance reliability and to optimize loss rates.

1.7 Thesis Outline

In Chapter 2, we have a look at related work required to understand this thesis. We will describe the multicasting paradigm, implementations of multicast, as well as different Peer-to-Peer and Application Layer Multicast (ALM) protocols. Besides giving an overview, we focus on the different protocols relevant for this work.

The Quality of Service for Overlay Multicast Framework "OM-QoS" is presented in Chapter 3. OM-QoS aims to enable QoS guarantees with multiple QoS aspects (bandwidth, end-to-end delays, loss rate, hop-by-hop jitter, etc.) for various P2P/ALM networks and application scenarios. First, the structure of QoS supporting multicast trees are described. Then, the solutions that require some protocol specific modifications of the P2P/ALM protocols are presented as well as a protocol independent solution using layers. Our approach is able to support multiple hop-by-hop dependent QoS parameters aggregated into discrete QoS classes, while also supporting end-to-end delay requirements in the form of node to root RTT guarantees. The different approaches of OM-QoS and their application to the investigated protocols is evaluated using simulations.

There are many different P2P and ALM protocols. Each of the different protocols may be targeted at certain specific scenarios or use cases. Unfortunately, no standard for P2P/ALM exists as this is the case for IP Multicast. Therefore, it would be desirable to be able to profit from both IP Multicast and ALM together in a combined manner. In Chapter 4, we describe the Multicast Middleware, our solution for providing IP Multicast services via P2P / Overlay Networks. It facilitates using the IP Multicast API while data is though transmitted using ALM mechanisms. This is done in a completely transparent way for applications. The presented solution enables end users to use IP Multicast functionality through the Internet without requiring any IP Multicast support by the network and routers. Hence, users can benefit from widely available applications supporting IP Multicast while not being limited by the lack of a global deployment of IP Multicast. After a presentation of the design and implementation, we show different evaluations that have been performed to determine the performance of the Multicast Middleware.

Our solution for efficient data dissemination using cooperation with the help of multicasting called "MCFTP" (Multicast File Transfer Protocol), is presented in Chapter 5. We first explain how MCFTP works and then compare it to BitTorrent. MCFTP enables efficient usage of already downloaded sub parts of files for direct retransmission while also benefiting from the multicast paradigm for efficient network communication. Using a simulator, MCFTP is evaluated and compared with BitTorrent in terms of performance regarding download time, bandwidth usage, etc. Also, a prototype implementation is presented, which has been evaluated and compared with the popular BitTorrent implementation Azureus. The evaluations have been performed using IP Multicast and Overlay Multicast scenarios. These scenarios have been evaluated in a local testbed.

Chapter 6 concludes this thesis with a summary, conclusion and outlook.

Chapter 2

Related Work

2.1 Introduction

In this Chapter, a general introduction to Multicasting, Overlay Networks, and Quality of Service (QoS) concepts is given. We focus on topics relevant for better understanding this thesis. Overlay Networks are often also called Peer-to-Peer (P2P) networks. Multicasting using Overlay Networks is often referred to as Overlay Multicast or Application Layer Multicast (ALM).

The presented protocols and concepts in this Chapter represent only a subset of all available related systems. There exists a wide variety of P2P/ALM protocols and concepts with various characteristics and features. We have chosen the ones that are important to understand the contributions of this thesis and that represent mixed characteristics of P2P/ALM concepts and protocols. Some of the presented protocols have been extended, improved or were used to compare our contributions with. We also present some early work done in the field of P2P/ALM research.

Multicasting is an efficient mechanism for a sender to distribute data to many receivers concurrently. Unfortunately, IP Multicast (the multicasting implementation for the Internet) is not widely available to the end user in the Internet today.

P2P networks solve different problems of classical client-server approaches, such as scalability, reliability, cost-efficiency and many more. There is no distinction in P2P networks between servers and clients. Every peer can be server, client or both at the same time. P2P networks are usually self-organizing which means that they do not require any special infrastructure support. They also adapt very well to changing conditions, such as leaving/joining of peers, network failure, etc.

ALM, Overlay Multicast or End-System Multicast offers multicasting functionality to end users independent of the availability of IP Multicast. ALM protocols often run on-top of existing P2P Overlay Networks, in order to offer an efficient and reliable multicasting service.

To give the best service to end users, QoS mechanisms for P2P/ALM networks would be desirable. This enables enhancing user experience for applications such as IP-TV, multiplayer online games, etc. The introduction of QoS mechanisms for P2P/ALM networks is one of the goals of this thesis.

Section 2.2 gives an overview of the multicasting paradigm and different implementations, such as IP Multicast and Overlay Multicast. In Section 2.3, we present an overview of different P2P networks, look at the relevant protocols for this thesis, and present some P2P applications in Section 2.4. An ALM overview with a more detailed analysis of the relevant protocols is given in Section 2.5, and some ALM applications are presented in Section 2.6. Quality of Service and solutions to support QoS for P2P/ALM networks are presented in Section 2.7.

2.2 Multicasting

2.2.1 Overview

Multicasting [115] is an efficient mechanism for a sender to distribute, e.g., multimedia data to many receivers concurrently. Multicast communication enables delivery of data (such as audio or video streams) from one sender to multiple receivers (a group of hosts interested in receiving the data) with minimal network overhead. With unicast communication, the sender must send the data separately to each receiver.

Using the multicast communication paradigm in the Internet, enables replication of data packets by the transporting routers or in end systems only when needed. In this way, the network load is minimized – the data flow traverses each link at the most optimal case only once.

2.2.2 IP Multicast

The Internet Protocol (IP) [133] is designed with built-in support for multicast [60, 72, 48] communication. IP Multicast has been proposed and specified two decades ago. It reduces network load by eliminating the redundancy of data transfer by replicating data in routers only when required. It is a concept for efficient n-to-m data dissemination over IP networks.

Basically in IP Multicast, the sender sends an IP datagram to a so-called group address. Such an address is a special IP address from a predefined range [6, 91] reserved for use as multicast group addresses. The IP datagram is forwarded to all receivers interested in receiving the data by replicating the IP datagram on the path to the receivers only when needed. Therefore, all hosts, which are interested in receiving data that is sent to a multicast group address, must subscribe to that address. An IP packet sent to a multicast group address is forwarded to all subscribed hosts for this group. To achieve this, IP routers in the Internet must have information over which network interfaces an IP packet with a destination address of a multicast group has to be forwarded. This information is exchanged between all IP routers, which may be involved in the forwarding process.

On a global scale, routing information for one multicast group is represented as a so called multicast tree. The root of a multicast tree is the sender of the corresponding IP Multicast packets. Routers that are involved in transporting multicast

traffic are represented as nodes in this multicast tree and receivers are represented as leaves. Upon reception, routers in this multicast tree replicate IP Multicast packets to their children nodes. By doing so, IP routers minimize the overall network load compared to using unicast transmission where the sender sends the same IP datagram once for each receiver.

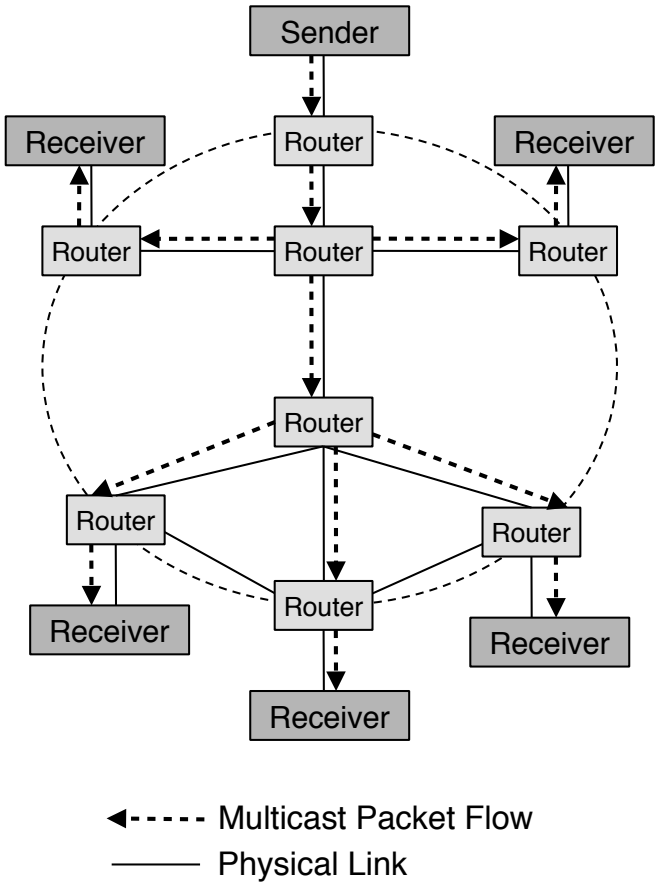


Figure 2.1: IP Multicast Distribution using Routers to Replicate Data

An example of a typical IP Multicast tree with one sender and multiple receivers is presented in Fig. 2.1. IP Multicast data is replicated at routers on the path from the sender to the subscribed end systems. The sender only sends the information once and routers take care of replicating the information where needed. There is no redundancy of data transport through the physical network links. In this way, the network load is minimized compared to unicast transmission. Using unicast, the sender would have to send the same IP datagram once for each receiver.

Unfortunately, even today, IP Multicast has not been widely deployed in networks of commercial Internet service providers (ISPs). Some reasons [61, 154, 63] for this are:

- IP Multicast must be supported by all routers on the path from source to destination.
- Additional inter-ISP coordination is required (policy issues of inter-domain routing).
- IP Multicast routing can be very resource intensive.
- Security concerns have to be addressed.
- Charging and inter-provider billing issues have to be solved.

2.2.3 MBONE

As a transition between the Internet without IP Multicast and its full availability to the end user, the MBONE [64, 147] approach has been proposed. In MBONE, the Internet is considered as a set of isolated IP Multicast enabled "islands". These "islands" are interconnected by an overlay network of tunnels. The overlay network is used to tunnel IP Multicast traffic between MBONE islands over parts of the Internet that support only unicast traffic. MBONE tunnels are implemented using the loose source routing (LSRR IP option) or by encapsulating IP Multicast packets in unicast packets. The drawback of this approach is that those tunnels have to be set up manually. As a consequence, the tunnel end-points must be permanently available and require fixed IP addresses.

This prohibits most Internet end users from using MBONE, since usually, end users are not permanently connected (e.g., using a modem). Normally, end users also do not have fixed IP addresses assigned when they are using modem, cable or xDSL connections. Furthermore, most of the typical Internet users can not handle the administrative overhead and coordination requirements of MBONE.

2.2.4 Overlay Multicast

Since MBONE was not able to provide multicast communication to end user in the Internet, numerous solutions were proposed to address this problem. The rising popularity of P2P [10, 108] networks lead to a revival of multicast in the form of ALM [86], which is often also referred to as Overlay Multicast. ALM mechanisms use similar methods as IP Multicast for data dissemination, but move the task of replicating multicast data from routers to end systems.

The advantage of this approach is that ALM mechanisms do not require multicast support by the routers, the network, nor the operating system. On the other hand, the replication of data only on end systems is not as efficient as replication in routers. For example, Fig. 2.2 shows a typical ALM scenario, which uses only unicast communication between end systems to enable multicast services. In comparison to IP Multicast (presented in Fig. 2.1), there is some redundancy of data that is sent over physical links.

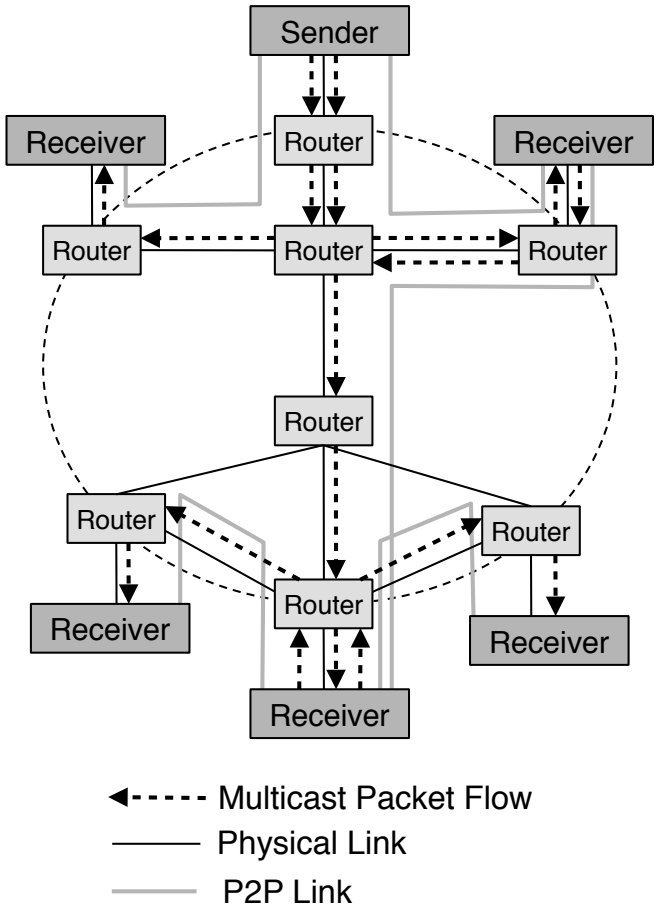


Figure 2.2: Application Layer Multicast Distribution using End Systems

The reason for this is that replication of data is only done on end systems and not on routers. Although ALM mechanisms can never achieve the efficiency of IP Multicast regarding the usage of network resources, they are still much more efficient than unicast communication between a sender and all its receivers. The efficiency of Application Layer Multicast depends on the overlay network construction and routing. With an optimal overlay topology, Application Layer Multicast can approximate the efficiency of native IP Multicast.

The advantages of ALM mechanisms are that they are designed to be self-organizing and fault-tolerant, which make them easier to deploy than MBONE. This makes ALM mechanisms better candidates for deploying multicast services to end users. The drawback of using ALM mechanisms is that the protocols used and APIs are not standardized, which makes application development dependent on specific ALM protocols. Another drawback is that existing IP Multicast enabled applications would have to be adapted to the specific API of ALM protocols.

2.2.5 Multicast Applications

IP Multicast Applications for Collaboration and Video Streaming

Although IP Multicast is not widely available, there exist numerous applications using it. With the rise of MBONE, several applications have been introduced to make use of efficient multicast data distribution. The MBONE video conferencing tools for example include among others:

- vat (visual audio tool) [169, 170]
- nv (Network video tool) [123]
- vic (video conferencing tool) [171, 172]
- NeVoT (network voice terminal) [120, 121, 151]
- wb (white board) [178]
- INRIA Videoconferencing System (ivs) [93]

Many other IP Multicast applications exist [115]. The Access Grid Project, for example, offers the Multicast Application Sharing Tool (MAST) [103, 110]. Microsoft research developed the advanced collaboration and interactive distance learning software called ConferenceXP [56]. This software also uses IP Multicast to achieve an efficient communication between the collaborating parties. Video LAN Client (VLC) [174] is an IP Multicast enabled video broadcasting and video playback tool, which we also used to test and evaluate our Multicast Middleware.

Virtual Environments, Massively Online Multiplayer Games and Collaboration Tools using Overlay Multicast

There are also many other application scenarios, which can benefit from multicasting. These scenarios are though often based on Overlay Multicast, each having their own application specific optimized multicast implementation.

Such application scenarios include collaboration tools for audio/video conferencing and concurrent document editing. An end system multicast protocol for collaborative virtual environments has been presented in [87]. The authors propose their own end system multicast protocol for multi-sender virtual teleconference applications. A controllable audio/video collaboration system based on multicast is presented in [187, 186]. The presented system has been implemented and applied on the China education and research network CERNET. Momocomm [117] is an implementation of a multi-objective optimization based multicast overlay communication primitive, which can be used for P2P real-time collaboration tools.

Other scenarios could include Network Virtual Environments (NVEs), such as massively multiplayer online games (MMOGs), where multicast groups could be formed among party members, local regions in the game, different chat channels,

guild members, etc. Multicast reflectors for multiplayer online games have been proposed in [18]. Furthermore, a general communication architecture, which covers centralized, distributed and hybrid architectures has been presented in [17]. The architecture uses, depending on the application scenario, either client-server, P2P or a federated P2P approach. A hybrid MMOG architecture called MM-VISA (Massively Multiuser VIrtual Simulation Architecture) has been presented in [5]. The virtual world is decomposed into smaller manageable zones, which is then reflected into according P2P networks. There are also libraries available to support scalable P2P NVE applications and MMOGs. VAST [168] is such a light-weight network library, which is based on a Voronoi-based Overlay Network (VON) as presented in [89, 88]. But, all these applications normally use their own specialized implementation of Overlay Multicast tightly integrated into the applications. They usually do not use existing multi-purpose ALM frameworks.

2.2.6 Communication Restrictions for Multicasting

Besides the lack of global IP Multicast support, there are further limiting factors for communication in the Internet. The most severe impact on the communication limitations come from Firewalls and Network address translators (NATs). These devices restrict the ability of a host to open connections to hosts in other networks.

Firewalls are network devices that filter portions of network traffic based on protocol header information and/or data payload of IP packets. They are used to protect network devices within a private network from intrusions from the Internet. Firewalls normally limit the ability of hosts in the Internet to connect to hosts behind them. In some networks, they are also used to prevent communication of hosts behind the firewall with hosts in the Internet.

Such installations are used as a preventive measure against Trojan horses, Internet worms, etc. This also limits the use of P2P/ALM applications on hosts behind firewalls, since these applications assume universal connectivity between hosts. Firewalls also have to handle and take care of the special nature and threats arising of relaying IP Multicast traffic [73].

The address space of the IP protocol version 4 is limited to 2^{32} addresses. In addition, there are IP address ranges that cannot be used for end host addresses, such as the address ranges reserved for private use, the IP Multicast address range, the network addresses, and the broadcast addresses. To overcome this limitation, NATs were introduced.

NATs are network devices, which allow a whole private network to appear as one IP address on the Internet. NATs achieve this by modifying the outgoing traffic to appear to be originating from the same IP address. The incoming traffic is reverse-mapped to its corresponding destination in the private network. The mapping of public IP address port numbers to private address port numbers and the IP address/port pair is stored inside the NAT. Traffic incoming from the Internet is translated to the private addressing scheme according to the stored mapping and forwarded to the private network. In this way, a NAT provides Internet connectiv-

ity for all host in a private network using only one public IP address visible from the Internet. The drawback of using NATs is that P2P/ALM communication is affected, since end systems behind a NAT are not able to accept any incoming connections. There are several proposals for solving this problem such as UPNP [165]. Nevertheless, no universal solution for this problem is available currently. Another issue with NATs is that the idea of IP addressing, where each host has a unique address, is violated. The hosts behind a NAT usually have an IP address from a range of IP addresses that are reserved for private use (10.0.0.0/8, 192.168.0.0/16 or 172.16.0.0/20). This means that there are potentially many hosts with the same IP address. This aspect of NATs also subverts different P2P node ID generation schemes, which assume that every host has a unique IP address.

All communication obstacles presented also apply to IP Multicast. In reality NATs and firewalls are widespread and restrict the connectivity of the endpoints. For an in-depth analysis of connectivity restrictions for overlay networks see [76].

In this thesis, we will not consider connection restrictions, because they are not our main focus. Nevertheless, these problems are nowadays also starting to be addressed by P2P/ALM networks. Therefore, choosing an ALM for our proposed solutions in this thesis that addresses the mentioned issues, reduces the impact of connection restrictions on our presented work.

2.3 Peer-to-Peer Networks

2.3.1 Overview

The use of Peer-to-Peer (P2P) [10, 108] has become very popular over the past few years. Different Peer-to-Peer (P2P) architectures [9, 143, 188, 136, 161, 160, 185] exist. Each of them have their own advantages and disadvantages depending on the application they are targeted at.

The authors of [10] give a survey of Peer-to-Peer content distribution technologies. A framework for analyzing peer-to-peer content distributing technologies is presented. It focuses on nonfunctional characteristics like security, scalability, performance, fairness and resource management. Studies about routing mechanisms, applied distributed object location mechanisms, content replication, caching, migration and security related issues are also performed in that paper. The security issues include encryption support, access control, authentication, etc. In the authors perspective, there exist two defining characteristics of Peer-to-Peer architectures. The first is sharing computer resources by direct exchange. The second is treating instability and variable connectivity as the norm. They propose a classification of peer-to-peer applications. This is done in regard of communication and collaboration, distributed computation, Internet service support, database systems and also content distribution. Furthermore, they investigate the overlay network centralization and classify it into purely or partially centralized and hybrid decentralized architectures. Also, the different overlay network structures are classified into different categories of overlay networks.

In [108], a survey and comparison of various P2P overlay networks is presented. The different overlay network schemes investigated are categorized into these two groups. The design of the different overlay network protocols is analyzed regarding various criteria. These include look-up protocol, system parameters, routing performance, security, reliability / fault resiliency, and many more. Their final thoughts are about directions for the future in P2P overlay networking research. This includes concerns about mapping the virtual topology of P2P networks to the underlaying physical network infrastructure. They also plead a case for introducing some sort of incentive model using economic and game theories for encouraging peers to collaborate. Also trust and reputation is deemed important for enabling secured and trustworthy P2P overlay communication.

A Resilient Overlay Network (RON) [9, 139] presented in Section 2.3.2 provides an architecture to improve robustness and efficiency of communication for distributed applications. Nodes in RON monitor and probe paths in the overlay network. By exchanging the monitored and probed information with other RON nodes, they can decide to either use direct routing offered in the Internet or route via other RON nodes to reach the destination. Hence, RON supports application-specific metrics for routing and can provide alternative routes to reach a destination instead of relying on default routing mechanisms of the Internet.

Pastry [143, 51] presented in Section 2.3.3 is a scalable distributed object location and routing substrate for P2P applications. In Pastry, peers assign themselves a randomly chosen ID when they join the P2P network. To route a message to a certain Peer, Pastry uses an efficient routing algorithm, which scales logarithmically with the number of peers in the P2P network. Pastry is self-organizing and completely decentralized. It also takes peer proximity information (in terms of end-to-end delay) into account to minimize the distance messages are traveling.

Tapestry [188] is mainly used for distributed data storage. Therefore, it provides a location and routing infrastructure to peers participating int the overlay network. Data objects stored at specific locations in the overlay are advertised and replicated at other peers. Look-up messages to find specific objects are routed trough the overlay to the closest copy of a stored object. Load at the object's location is reduced due to advertisement caching and object replication. The main features of Tapestry are self-administration, fault tolerance, and resilience under load. In Section 2.3.4, more details about Tapestry will be presented.

Content Addressable Networks (CAN) [137, 136] presented in Section 2.3.5 use a virtual d-dimensional Cartesian coordinate space to store key - value pairs. The space is partitioned into n (number of hosts) zones, which correspond to the keys. Each node is responsible for managing one part of the coordinate space. The coordinate space has to be rearranged if new nodes join, existing zones have to be divided between the new and existing nodes. Changes of zones are propagated to neighbors (adjacent zones), who update their neighbor sets. Neighbor sets contain all information about adjacent zones for a host. Routing is greedy using the neighbor closest to the destination as next hop. Multicast is done by sending a message to all the neighbors of a host with a duplicate suppressing mechanism.

Chord [161, 160] is a Distributed Hash Table (DHT) [97] based structured P2P network. It offers efficient location of the node that stores a desired data item. Chord uses a distributed look-up protocol for object location. Chord is very scalable and simple in its architecture. The communication costs and states maintained by nodes scale logarithmically regarding the number of Chord nodes. Chord uses successor lists and finger tables to optimize routing in the overlay network. The overlay structure is periodically optimized using stabilization, which refreshes or updates finger table entries and successors incrementally. In Section 2.3.6, more details about Chord will be presented.

2.3.2 RON

A Resilient Overlay Network (RON) [9, 139] provides a robust and efficient communication architecture for distributed Internet applications by building an overlay network. RON supports detection and quick recovery (in terms of seconds) from path outages. To achieve this, RON nodes monitor paths of the overlay network that they built among themselves. Using this monitoring information, RON nodes can decide to either route packets directly over the Internet using default Border Gateway Protocol (BGP) [138] routing to reach the destination, or by using other RON nodes circumventing BGP policy routing. This facilitates more efficient / robust routing and also supporting application-specific routing metrics. RON can bypass link failures by often only using one intermediate hop. Nodes in RON share the monitored and probed information among each other. The monitoring and probing metrics include packet loss rate, available throughput, and latency.

RON nodes are distributed over different routing domains and build an overlay network in order to forward data on behalf of any pair of RON nodes that wants to communicate with each other. Because of the nature of the Internet, routing domains very seldom share interior links and generally have failures independently of each other. Hence, RON often finds paths between its nodes if the underlying topology has physical path redundancy, even when BGP fails to do so.

RON includes several components: overlay configuration / maintenance, probing / outage detection, routing around outages / performance failures, application-controlled / policy / multi-path routing, data forwarding, BGP interactions, etc. Applications using RON include resilient VPN, resilient video conferencing, etc.

2.3.3 Pastry

Pastry [143, 51] is a P2P location and routing substrate with a ring structure (one-dimensional torus). Each peer is identified by a 128 bit long ID. This ID is randomly chosen when joining a Pastry network. The choice of IDs is uniformly distributed. As a consequence, the choice of the ID neither takes locality nor QoS requirements into account.

Each Pastry peer has a routing table with the size of $(2^b - 1) * \lceil log_{2^b} N \rceil + l$ entries. The routing tables are organized into $\lceil log_{2^b} N \rceil$ rows with each $2^b - 1$

entries. The entries of row n of a peer's routing table point to other peers, which share the same first n digits of their ID with the peer itself. But, the digit at position $n+1$ has one of the 2^b-1 possible values different from the digit at position $n+1$ of the peer's ID. Each entry in the routing table consists of the destination's ID and its corresponding IP address. Additionally, each peer maintains a list of numerically closest peers (IDs and IP addresses) with $l/2$ entries for the larger and $l/2$ entries for the lower IDs. Locality is taken into account by choosing the closest peer (in terms the network latency) among candidates for an entry in the routing table. A message is routed to the closest neighbor found in the peer's routing table whose ID matches the messages destination ID prefix.

Routing uses less than $\lceil log_{2^b} N \rceil$ steps on average, where N is the amount of peers in the pastry Network and b is typically a parameter with the value 4. Pastry guarantees eventual delivery of a message unless $l/2$ or more peers with an adjacent ID fail at the same time, with l, an even number parameter, being typically 16. Routing in Pastry uses Plaxton's method [132]. Each hop from source to destination tries to match one or more prefixes of the message's destination address. A peer has more information about ID-neighbors (matching many prefixes of the ID left to right) than about ID-distant peers matching less ID prefixes (left-to-right). Routing in Pastry is proximity aware. Each hop from source to destination tries to minimize the intermediate hop-delay. Therefore, the overall end-to-end delay can be optimized up to a certain degree.

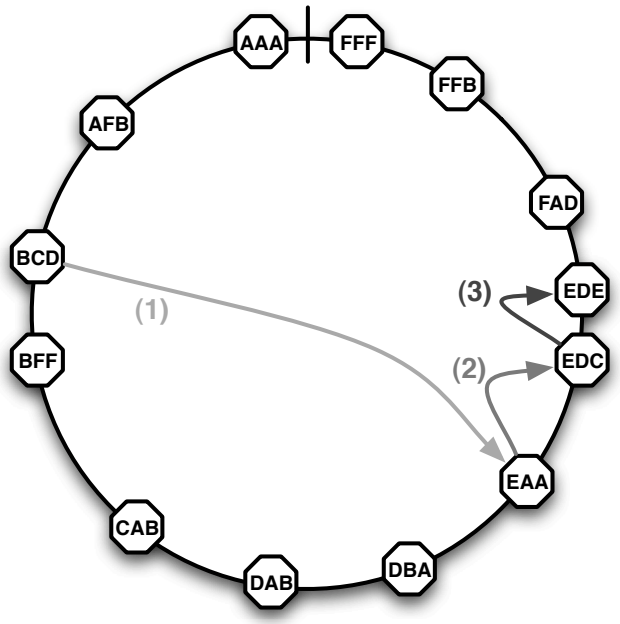


Figure 2.3: Routing a Message using Pastry from Node BCD to Node EDE

Figure 2.3 shows a simplified example of Pastry's prefix matching routing mechanism. In the example we want to route a message from the source *BCD*

to the destination *EDE*. The source only knows the node *EAA*, which shares the first prefix with the destination address *EDE*. The message is forwarded (1) to this node, which forwards (2) the message to *EDC* matching the next prefix of the destination address in its own routing table. Finally, node *EDC* delivers (3) the message to the destination *EDE*, which it knows directly. If there is no node, with the ID of a message's destination address assigned, the node with the ID numerically closest to the destination address is responsible for message message.

2.3.4 Tapestry

Tapestry [188] is similar to Pastry as described in Section 2.3.3 and uses also prefix routing as described before with randomly assigned IDs. The prefixes are matched from right-to-left, whereas Pastry uses left-to-right prefix matching. Figure 2.4 shows a simplified example of Tapestry's routing mechanism. In the example, a message is sent from node *8311* to node *4985*. On each hop, one additional prefix is matched (xxx5 $\rightarrow$ xx85 $\rightarrow$ x985 $\rightarrow$ 4985).

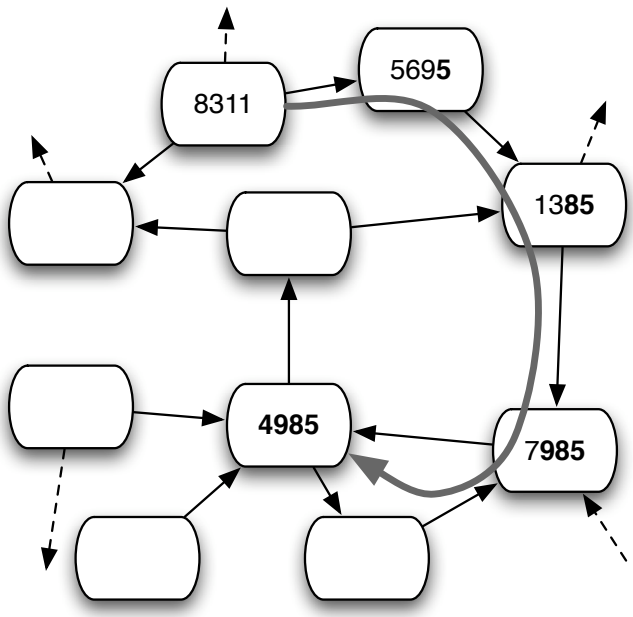


Figure 2.4: Tapestry Routing from Node 8311 to Node 4985 with Prefix Matching

Tapestry is mainly used for distributed data storage. An example is shown in Figure 2.5. A data object O stored at S is advertised sending a *publish* message to N (whose $ID = hash(O)$) using Tapestry's routing. Intermediate nodes on the path cache this advertisement. Nodes requiring the location information of object O can find and contact N. The do this using the hash function $hash(O)$ but might already get an answer from a node on the path to N caching the requested information. Caching reduces the load on the look-up node N.

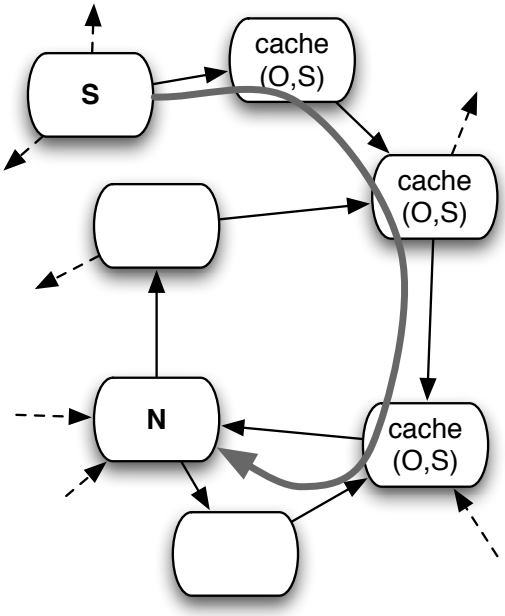


Figure 2.5: Server S Advertises his Object O on Node N using Tapestry

2.3.5 CAN

CAN [137, 136] P2P networks are built using a virtual d-dimensional coordinate space (d-torus). Each new node joining the CAN P2P network assigns itself some random coordinates in the virtual space. It then finds an existing node responsible for the sub-part of the space, in which the new node's coordinates lay. The node responsible for that sub-space then splits its sub-part of the space in half and assigns one half to the new node and keeps the other part to itself. Responsible nodes for the adjacent zones to the newly created sub-parts (neighbors) are then updated with the new information.

An example of a join sequence in a 2-dimensional CAN is described in Fig. 2.6. The first node assigns itself the coordinates (0.8, 0.6). Since no other node is yet inhabiting the CAN space, this first node becomes responsible for the full coordinate space of CAN [0-1),[0-1). The second node joining the network assigns itself the coordinates (0.3,0.4). The first node then splits its responsible space in half and assigns the part, in which the new node's coordinates are located to the new joining node. Therefore, the second node is now responsible for the zone [0-0.5),[0-1), whereas the first node now is only responsible for the zone [0.5-1),[0-1). The third node joining with coordinates (0.2,0.9) triggers the second node to split its zone. The second node's new zone is set to [0-0.5),[0-0.5) and the new joining node becomes responsible for the zone [0-0.5),[0.5-1). Finally, a fourth node with the coordinates (0.4,0.7) joining CAN becomes responsible for the zone [0.25-0.5),[0.5-1). The splitting third node is now only responsible for the zone [0-0.25),[0.5-1).

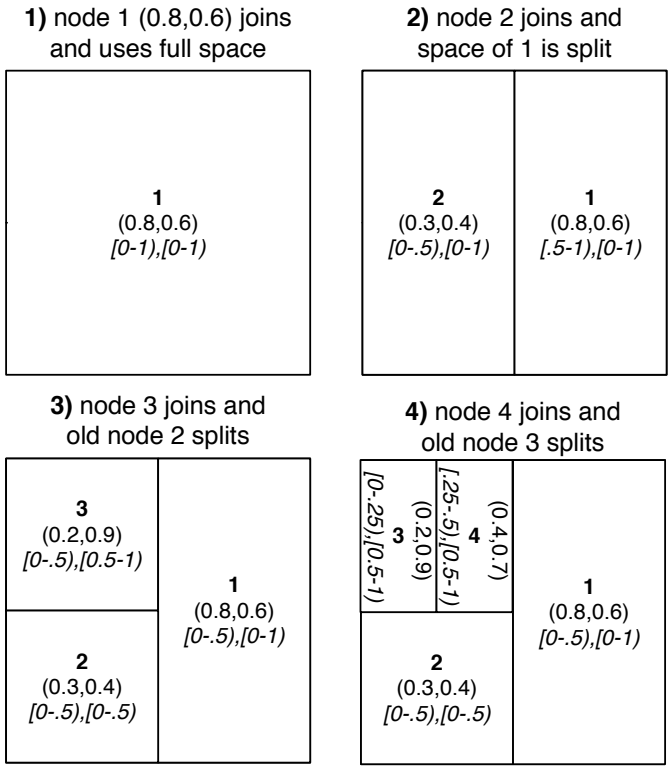


Figure 2.6: Nodes Joining a CAN

Routing in CAN is very simple. It uses a greedy routing that tries to pass via the next neighbor closer to the destination coordinates. Figure 2.7(a) shows a sample routing from node 1 to node 9 with the intermediate hops 6 and 11. In

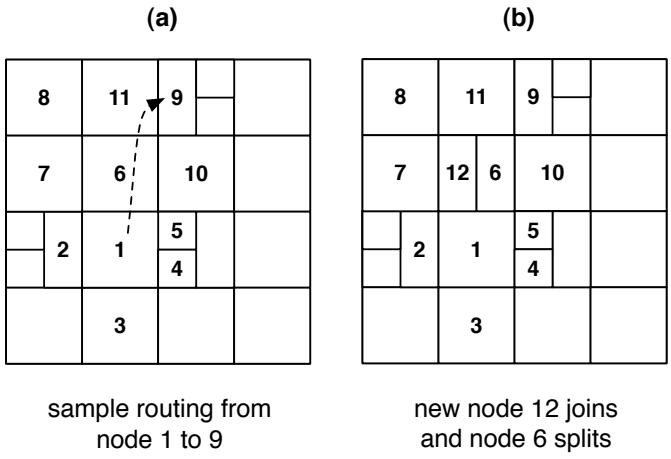


Figure 2.7: Routing in CAN

this example, node 1 wants to route a message to the coordinates (0.55,0.95). It looks at its neighbor set (2,3,4,5,6) and finds node 6 being closest to the destination coordinates, and hence forwards the route request to node 6. Node 6 then consults its own neighbor set (1,7,10,11) and forwards the route request to node 11, which would be closest node to the destination coordinates. Finally, node 11 knows that the destination coordinates of the route request are in the zone, for which its direct neighbor node 9 is responsible. Therefore, it forwards the message to its final destination.

Splitting zones always requires that the neighbors are being updated about the new situation. This is shown in Fig. 2.7(b) with node 12 joining the CAN network. Node 6 would split its own zone and change its neighbor set to (1,10,11,12). Node 12 would build its own neighbor set (1,6,7,11) and inform nodes 1,7 and 11 about the changes. Node 7 then replaces its previous neighbor 6 with the new node 12. Nodes 1 and 11 add node 12 as new neighbor and reduce the responsible zone for node 6 in its neighbor table accordingly.

Finally, when a node leaves, it has to inform the neighbors and needs to find a node that can take over its responsible zone. A node might be responsible for more than one zone at a time and zones might be merged together as well.

2.3.6 Chord

Chord [161, 160] is a distributed hash table (DHT) [97] based structured P2P network. A node that joins a Chord network assigns itself a random ID in the range from 0 to n, with n typically being 2^{128}. Nodes in Chord have pointers to k successors. Each node also maintains a so called *finger table*. There, pointers to $log_2(n)$ nodes are stored to get a better knowledge about nodes distributed over the whole ID space. Periodically, a so called *stabilization* is performed which updates and repairs the finger and successor tables incrementally. Using stabilization, nodes can also confirm that they are the predecessors of their successors. Therefore, stabilization helps to keep the Chord ring intact. An example of a simple Chord network is presented in Fig. 2.8.

Message routing in Chord is simple. A node that wants to send a message to another node consults its finger table and successor list. If it finds the destination ID in any of those lists, it can directly forward the message to the destination. If the destination ID is between its own ID and the ID of the next node in its successor list, it assumes that there is no existing node at the destination ID. This can happen if a large ID space is used but only few peers actually participate in a Chord network. Therefore, it will send the message to its first successor, which would be responsible for the destination address. Otherwise, a node checks whether it has an ID entry in its finger table, which is close to the destination ID. The ID in the finger table also has to be lower than the destination ID. It then sends the message to that ID found in the finger table for further routing. If no entry has been found in the finger table, the node forwards the message to its successor being closest to the destination. Again, the ID of the successor has to be lower than the destina-

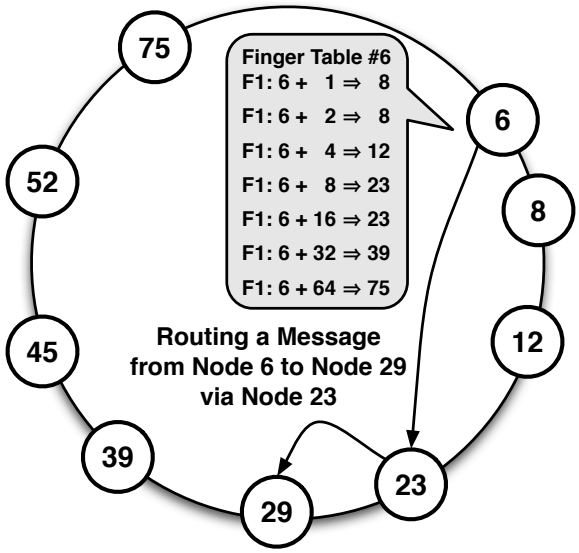


Figure 2.8: Example of a Chord P2P Network

tion ID. This forwarding process is repeated until the destination itself or the node responsible for the destination ID is reached.

The authors of [54] offer an experimental implementation of Chord. The Cooperative File System (CFS) [58] is an efficient and robust peer-to-peer read-only storage system based on Chord/DHash. Also Java implementations of Chord, such as Open Chord [125] or Chordless [55] exist.

In [59], an improved Chord model is presented. Nodes are organized into groups with the aim to enhance look-up efficiency. This should also decrease the impact of dynamic environments. The extensions also facilitate transmission of a shared file from multiple nodes simultaneously. This helps to decrease download times.

2.4 Applications of P2P Overlay Networks

2.4.1 Overview

There are many different applications, extensions, services, and frameworks, which are built on P2P overlay network structures. They are normally targeted at different specific application domains or scenarios. Section 2.2.5 already gave an overview of possible application domains and scenarios for multicasting. Here, we look at examples not directly related to multicasting. These different approaches are mainly targeted at efficient file distribution, data dissemination or streaming scenarios. Often, they are based on linear or tree topologies as presented in [21].

Content Distribution Networks (CDNs) presented in Section 2.4.2 help to distribute frequently requested data to strategic key positions in the Internet. The

participating reflectors are building an overlay network among themselves to efficiently distribute the required data.

Slurpie presented in Section 2.4.3 is a P2P protocol for bulk data transfer. It is used to reduce download times for users when downloading large files. A P2P network is established among the users downloading the same file.

Bullet is used for high-bandwidth data distribution and is presented in Section 2.4.4. A P2P mesh structure is used to efficiently disseminate the data from a single source to multiple receivers.

FastReplica presented in Section 2.4.5 is targeted at efficient and reliable replication of large files in the Internet. It uses an overlay network to distribute a large file among typically 10-30 nodes.

FTP-M presented in Section 2.4.6 is used for reliable multicast data file distribution in scenarios such as distributed backups, web-server mirroring, software update distribution, etc. It uses overlay structures using TCP-M for parallel data distribution of files.

BitTorrent builds an overlay structure in the form of a swarm among peers downloading the same file and is presented in Section 2.4.7. The swarm is normally coordinated by a tracker, and the peers decide by themselves about which other peers they want to interacts with.

GridFTP is used for reliable data distribution in high-bandwidth wide-area networks. This secure and high-performance protocol is based on FTP, the classical file transfer protocol of the Internet. More details about GridFTP are presented in Section 2.4.8.

The Application-Layer Traffic Optimization (ALTO) Working Group presented in Section 2.4.9 aims to provide solutions for different problems that have to be faced in P2P systems today. The Working Group presents a P2P communication protocol for applications and the ALTO servers.

2.4.2 Content Distribution Networks

For providing audio and video broadcasts for typical Internet users, content providers use content distribution networks (CDNs) [146]. A CDN consists of numerous hierarchically organized “reflector” hosts, which are receiving the content (audio and/or video stream) from hosts that are one level below in the hierarchy. Those “reflector” hosts redistribute the content to hosts one level lower in the hierarchy. The sending host is the highest host in the hierarchy and receiver hosts are located on the lowest hierarchy level. The “reflector” hosts are usually geographically dispersed all over the Internet to enable the end-user hosts to connect to reflectors, which are near them (in terms of the network latency).

The use of CDNs requires a substantial investment in infrastructure with costs rising according to the number of receivers. The communication protocol in CDNs is usually a proprietary protocol based on TCP or UDP. It requires special servers and clients for distributing and receiving the video stream.

2.4.3 Slurpie

Slurpie [157] is a Peer-to-Peer protocol for bulk data transfer. Slurpie reduces client download times for large files. It also lowers the load on servers that provide these files. Slurpie's design is scalable, easily deployable, adaptive to network conditions and compatible to existing protocols. The protocol of Slurpie itself is designed to be able to handle very large numbers of simultaneous clients and additionally lowers load on the server's side. Clients should only use Slurpie's download mechanisms when their own download time is reduced. Therefore, Slurpie tries to overall minimize download times needed by clients. Another goal of Slurpie is that it should be able to deploy it without any special infrastructure support.

Slurpie's design is also able to adapt to different network conditions. That means that it adapts its download strategy regarding the amount of available bandwidth and free processing resources at clients. It is not depending on changes to be implemented at the server-side. Slurpie can be used with existing data transfer protocols including HTTP and FTP. It offers an adaptive downloading strategy, which increases client's performance and uses a randomized back-off strategy to control the load on the server. Slurpie clients improve their downloading performance, if the size of the participants increases.

2.4.4 Bullet

Bullet [100] is targeting high-bandwidth data distribution using a single source. It supports data distribution to a large number of receivers. Bullet supports ordinary file transfers as well as real time streaming of data. It uses an overlay mesh instead of a tree. This way, it delivers fundamentally higher overall bandwidth according the authors of Bullet. This concept of using a mesh also should ensure better robustness and more reliability compared to traditional tree based structures.

Bullet offers a scalable and distributed algorithm based on self-organizing nodes. It supports distribution of data in a disjoint manner to the nodes in the overlay network. The receivers are responsible for locating and retrieving the data. They try to get data from multiple sources, such as the parent and other nodes in the overlay tree, in parallel. Bullet's distribution algorithm sends the to be disseminated data to different strategic key points in the overlay network. The locating algorithm of Bullet allows nodes to find and recover missing data items easily and fast. Nodes simultaneously receive data from multiple sources in parallel to speed up downloads. Therefore, it is not necessary to just find a single source that is able to provide all the data and which is capable of providing this data at high transmission rates. Relative to classical tree-based solutions, Bullet reduces the requirement of performing expensive bandwidth probing to find the optimal data providers.

2.4.5 FastReplica

FastReplica [53] is an algorithm for efficient and reliable replication of large files in the Internet environment using an overlay network. To distribute a large file

among n (n typically between 10-30) nodes, the original file is partitioned into n sub-files of equal size. Each file partition is sent to a different node in the overlay network consisting of all nodes interested in the file. Nodes then propagate their sub-file to the remaining nodes in overlay network. Instead of using n Internet paths, connecting the original node to the replication group (the overlay network), FastReplica uses $n \times n$ Internet paths. They are within the replication group, where each path is used for transferring $1/n$-th of the file. This way, FastReplica can be used for replication of large files to a group of nodes.

The use of FastReplica is simple and inexpensive. It does not require any changes or modifications to the existing Internet infrastructure. Experiments on a prototype implementation of FastReplica in a wide-area testbed show that FastReplica significantly reduces file replication time.

2.4.6 FTP-M

FTP-M [118] provides a reliable multicast data file distribution facility. It uses TCP-M [78], which is a "Multicast" deviate of classical TCP to ensure reliability. This also facilitates parallel data distribution of files. To fully use the introduced capabilities, modifications have to be done on the client as well as on the server side to the ftp-protocol and the applications used. The authors of FTP-M provide a convenient API as classical FTP enhancements for the BSD platform.

FTP-M offers two modes, classical FTP as well as the FTP-M extended version. The authors claim that FTP-M is reliable concerning the multicast data distribution. This is opposed to classical UDP based multicast, which does not guarantee delivery. FTP-M is mainly targeted for small or medium sized multicast scenarios. Typical usage scenarios would include distributed backups over the network, webserver mirroring, and software update distribution in cooperate environments.

2.4.7 BitTorrent

BitTorrent [26, 94, 180] is a P2P file sharing protocol for efficient data distribution that improves the download time of a file for users. It consists of a tracker and the downloading peers. The tracker holds information about participating peers in the swarm (P2P network). This includes how much of a file a peer already has downloaded and can offer those to other peers for download.

In BitTorrent, a file is split into sub-parts called chunks and has a coordination file called torrent file. This file stores information about the file (tracker, size, checksums, etc). Peers interested in a file have to retrieve the corresponding torrent file. This torrent file provides all the necessary information for nodes to join the "swarm" for that specific file. A swarm (using a P2P network) is built among the hosts that are interested in the same file. A tracker is then responsible to manage the swarm for a specific torrent file. With the help of the tracker, peers search and find other peers that already have the chunks the searching peer needs. Peers then exchange the missing chunks among themselves.

BitTorrent uses a tit-for-tat strategy to ensure cooperation among peers. A tracker can manage multiple torrent files. Also, peers can be part of multiple swarms. But one swarm is always dedicated to a specific file to be downloaded. There are never multiple files exchanged in one swarm.

2.4.8 GridFTP

GridFTP [7] is a protocol for efficient file transfers in grid environments. It has been proposed by the GridFTP Working Group [81] under the organization of the Global Grid Forum [126].

GridFTP extends the classical Internet File Transfer Protocol (FTP) [135] with new features, such as multi-streamed data transfers, Globus [79] based security, partial file transfers, authenticated and reusable transmission channels, and command pipelining. It is targeted at scenarios where huge amounts of data (terabytes) have to be transmitted efficiently and reliably. It is robust to failures, can parallelize data transfers (striping), offers extended security mechanisms, can give feedback to applications during transmission, and is easily extendable, which are all features that classical FTP is missing or only poorly supporting for high-bandwidth transmission in grid environments.

GridFTP uses TCP [134] for data transfers in grid environments. In wide area networks, the achieved data transfer throughput can fall below the actually available bandwidth due to the dynamic behavior of TCP. Hence, splitting TCP connection into segments can improve overall achievable throughput in high-bandwidth wide-area network environments. In [140], the authors present components to deploy overlay networks that use split-TCP connections in order to improve GridFTP's transfer performance. First, they evaluated using emulation which conditions can benefit most from splitting a TCP connection. Using empirical results, the authors demonstrate a significant performance improvement despite using intermittent, passive throughput observations to determine which proxies to choose.

2.4.9 ALTO

Solutions for different problems have to be faced in P2P systems today. The Application-Layer Traffic Optimization (ALTO) Working Group [8] aims to design and specify an ALTO service. This service should provide applications with information to perform better-than-random initial peer selection. The needs of BitTorrent, tracker-less P2P, and other applications, such as content delivery networks (CDNs) [146], and mirror selection will also be considered. A client will contact an ALTO server, of which it will get aware by using an already available discovery mechanism.

The WG will not propose standards on congestion signaling or avoidance, neither will it deal with information representing instantaneous network state. It will focus solely on the communication protocol between applications and ALTO

servers. The ALTO services offered by the ALTO server may be not only useful for P2P environments, but also in client-server environments.

The Working Group released various Internet Drafts. The ALTO requirements draft [99] looks at various requirements related issues, including ALTO client protocol requirements, ALTO server discovery, security considerations (high-level, classification of information disclosure scenarios, security requirements), privacy, rating criteria (performance-related, charging-related, distance-related), host group descriptors, and IANA considerations. The Alto protocol draft [98] describes the protocol, including the protocol structure regarding server capability and services, the network and cost map, protocol overview and messaging format, use cases, and IANA and security considerations. They also released another Internet Draft describing the problem statement, which has been updated to RFC 5693 [152]. It includes the definitions, the problem statement, example use cases (file sharing, cache/mirror selection, live media streaming, real-time communications, distributed hash tables), various aspects of the problem (provided information, service providers and implementation, user privacy, topology hiding, coexistence with caching), and security considerations. Finally, they will also prepare a draft for the ALTO discovery mechanism to be submitted as proposed standard.

2.5 Application Layer Multicast

2.5.1 Overview

Different protocols for ALM have been proposed [90, 144, 52, 49, 190, 15, 159, 184, 183].

In [86], different ALM protocols are analyzed and common design goals for ALM protocols are extracted. The authors look at different important properties of ALM protocol design, such as application domain, deployment level, group management, routing mechanisms as well as degree-constraints. They further differentiate between mesh-first and tree-first approaches. Different strategies for tree building and organizing are also explained. Finally different ALM protocols are compared according to the classifications introduced in the paper.

The authors of [69] look at the performance penalty of ALM over router level solutions. They compare three overlay multicast protocols regarding latency, bandwidth, router degrees, and host degrees. The comparison bases on experiments, simulations, and theoretical models. They also propose a way to quantify multicast overlay tree costs, defined as the total number of hops in all overlay links. They show that the number of hops between parent nodes and children nodes tends to decrease as the number of nodes in the overlay increases.

Narada [90] uses a two-step approach for building multicast trees in an overlay network built among end systems. First, a mesh is built that respects application requirements such as bandwidth or latency. Then, a spanning tree for each source is built on top of the mesh network. More details about Narada will be presented in Section 2.5.2.

Scribe [144, 52, 49] is a publish / subscribe system, which runs on top of Pastry and offers ALM functionality. It supports large groups, is fault-tolerant and decentralized. Scribe only provides best-effort reliability guarantees. It balances the load on nodes to reduce delays and lower the link stress. Scribe will be explained in more detail in Section 2.5.3.

Bayeux [190] is an ALM protocol that runs on-top of Tapestry. Multicast tree setup is performed using publish, join and tree messages. Opposite to Scribe/Pastry, it does not use reverse-path messages for tree setup. Section 2.5.4 will explain Bayeux in more detail.

CAN [137, 136] uses a flooding based mechanism to support multicast. In order to reduce the amount of duplicates caused by simple flooding, CAN introduces different duplicate reducing mechanisms, such as distance limits, duplicate suppression mechanisms and not forwarding messages along certain dimensions. Multicast in Chord will be described in Section 2.5.5.

Multicast in Chord [161, 160] can be realized using various approaches. The forwarder driven as well as the receiver driven multicast approach to support multicast for Chord will be presented in Section 2.5.6. Both approaches have their advantages and limitations. The forwarder driven approach can be easily implemented while the receiver driven approach is the only one that can be potentially used to introduce QoS with end-to-end delay guarantees.

NICE [15] is a hierarchically organized Overlay Multicast infrastructure using clusters and layers. It is locality-aware and tries to optimize latencies between cluster members. NICE is targeted at low bandwidth scenarios due to the high multicast fan-out of nodes. NICE will be explained in more detail in Section 2.5.7.

To reduce the fan-out, ZigZag [163], which extends NICE has been introduced. Basically, it eliminates the requirement for nodes to serve all layers where they are cluster leaders. Using ZigZag, the multicast fan-out of a cluster leader is independent of the numbers of layers. Section 2.5.8 will explain ZigZag in more detail.

VRing [159] uses a virtual ring as an Overlay Network among the multicast group members. To increase performance, a spare ring is built. More details about VRing can be found in Section 2.5.9.

Borg [184] is a scalable ALM for data dissemination in P2P networks. It looks at forward-path and reverse-path forwarding to optimize data transmission. Section 2.5.10 presents Borg in more detail.

PeerCast [183] is an effective passive replication scheme offering an efficient and self-configurable ALM framework. It takes proximity information into account to optimize data dissemination. More details about PeerCast are given in Section 2.5.11.

Dr. Multicast [173] is targeted at data centers. It offers policy controlled IP Multicast support using unicast and IP Multicast connections. Section 2.5.12 will present more details about Dr. Multicast.

2.5.2 Narada

In Narada [90], an overlay network is built directly among end systems in order to offer multicast functionality. It was one of the first protocols proposed for Overlay Multicast in end systems that has also been evaluated in the Internet. Narada is often also referred to as End System Multicast (ESM). Hence, it relies on end systems to support multicast communication rather than using IP Multicast services that require infrastructure support. It is also much easier to introduce new features and higher level functions regarding, e.g., flow control and reliability on the application level than trying to deploy new functionality in the network layer (IP Multicast) which often would result in infrastructure updates. The first evaluations of Narada / End System Multicast have been performed in simulators as well as distributed over the Internet using a variety of multicast applications. Narada is targeted at applications such as virtual classrooms, A/V conferencing, and multiplayer online games, all with typically tens to hundreds of group members.

The efficiency of the overlay network topology in Narada is optimized by end systems by adapting to dynamic changes in the underlying network and depending on application performance requirements. The topology is adapted to reduce redundant transmission and to support certain latency or bandwidth requirements of end systems. The Narada protocol is fully decentralized and the overlay is self-managed by the end systems. It is resilient to failures of end systems and to the dynamics of group memberships. End systems monitor passively and actively the characteristics of the overlay network links between them. This facilitates quick adaption of the overlay structure to changing network conditions in order to provide a reliable and well performing multicasting service.

Multicast trees in Narada are using a two-step setup process. First, a mesh is built among the end systems. Then, spanning trees are built on top of the mesh with each source as root of the corresponding spanning tree. This approach enables multi-source multicasting quite easily and also ensures optimal performance for multicast distribution from the different sources. The quality of the spanning trees depends heavily on the quality of the mesh. Hence, links between end systems in the mesh should be optimized according to the quality requirements of applications (e.g., bandwidth or latency). To build the trees on top of the mesh, a variant of the distance vector based algorithm is used. Furthermore, algorithms to build per-source (reverse) shortest path spanning trees are also applied.

Group management in Narada is shared among the participating end systems. Since Narada is only target at medium sized groups, the approach of the authors was to have all end systems be aware of all group members in order to support a high degree of robustness. These member lists at the end systems need to be constantly updated due to members joining or leaving. This is handled by sending periodical refresh messages with monotonically increasing sequence numbers to the neighbors in the mesh. To avoid mesh partitioning when a group member fails, group members store the last time they received a refresh message from a neighbor. After a certain period of time without receiving such messages, the potentially

failed group member is probed and either removed from the members list or new links to the member are added to repair the mesh.

2.5.3 Scribe

Scribe [144, 52, 49] is a publish / subscribe system, which runs on top of Pastry and offers a scalable and core based ALM infrastructure. Topics can be created and subscribed to. Every message that is published to a topic is sent to the corresponding subscribers. Topics in Scribe are used to provide ALM functionality. Any Scribe node can join any multicast group (or topic in Scribe's terminology) at any time. For each topic, one node is designated to disseminate the topic's data in the Pastry network. All multicast traffic for that topic is forwarded to the root node for dissemination.

Scribe offers best-effort delivery of the multicast data without guaranteeing that the order of the packets is maintained. Scribe builds on top of Pastry's routing and uses reverse-path forwarding trees to multicast messages. Each multicast group is represented by a topic ID. The host numerically closest to the topic ID becomes the root of the multicast distribution tree. All multicast messages are directly sent to this root node, which then multicasts the messages to all group members. To join a certain topic, a Scribe node sends a join message through the Pastry network. As explained previously in Section 2.3.3, the message is routed to the root node using prefix matching. Each Scribe node visited on the path also joins the same topic and remembers, which direct child nodes have subscribed to this specific topic ID. If a node already has joined the same topic ID earlier, no additional join message will be sent towards the root node. Finally, when messages have to be multicast to all group members subscribed to a specific topic ID, the root node sends the message to its direct children. These children then relay it to their direct children respectively. This is repeated until such a node has no more direct children to be served with the multicast message for the specific topic ID.

Figure 2.9 shows a simplified example of a multicast tree construction with Scribe. A joining node sends a *join* message to the topic's root node using Pastry's routing mechanism as previously shown in Fig. 2.3. Nodes on intermediate hops along the path of the *join* message add the previous node/hop to the list of receivers for that topic. A *join* message is only forwarded further towards the root if the current intermediate hop is not yet subscribed to that topic. When data for that topic ID has to be disseminated, the root node forwards the message to all its one-hop subscribed nodes. These nodes repeat the same process, forwarding the message to their one-hop subscribers. In the presented example, node *BCD* wants to join topic with ID *EDE*. The same intermediate hops are visited as in the Pastry example from Fig. 2.3. Therefore, the multicast tree distribution tree for this subscriber is built using the reverse path of the *join* message. Other joining hosts (e. g. *AFB* and *DAB*) could send their join message via nodes that are already subscribed to the topic (e. g. *EDC* and *EAA*). These nodes then stop forwarding the *join* message towards the root.

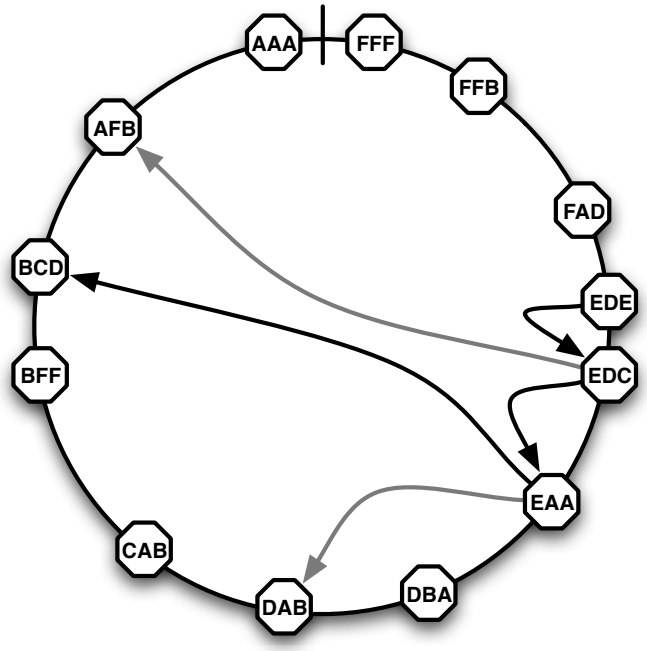


Figure 2.9: Scribe Topic Data Dissemination

2.5.4 Bayeux

Bayeux [190] is a source-specific, explicit join multicast facility, which runs on top of Tapestry. Figure 2.10 shows a simplified example of Bayeux's multicast tree creation. The root for a multicast group advertises, using a *publish* message,

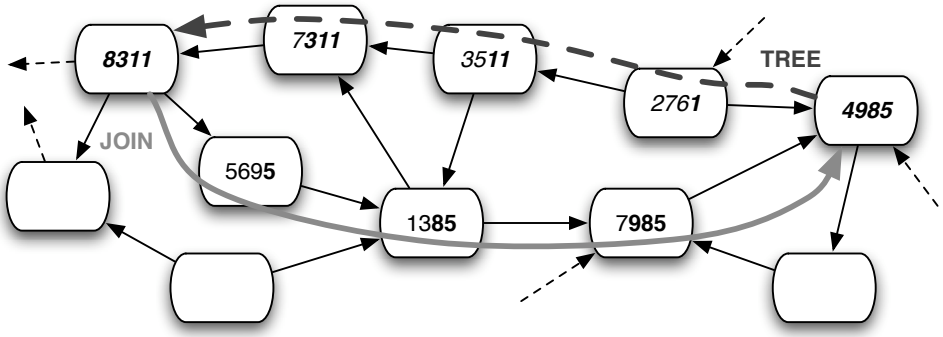


Figure 2.10: Multicasting using Bayeux for Tapestry

that it is the responsible node for this multicast group. Joining nodes send a *join* message to the root. The root node answers with a *tree* message. Each node on the path from the root to the joining node (for the *tree* message) saves the forwarding state $< dest, nexthop >$. The *join* messages are always delivered to the root node,

which results in higher link stress on the root compared to Scribe presented in Section 2.5.3. The multicast delivery path (constructed by the *tree* message) is not the reverse-path of the *join* message (as for Scribe).

2.5.5 Multicast in CAN

Multicast support in CAN is straight forward and very simple to implement. It is a flooding based mechanism, which has been further enhanced in order to avoid duplicates. Simple flooding in CAN is done by forwarding a multicast message to

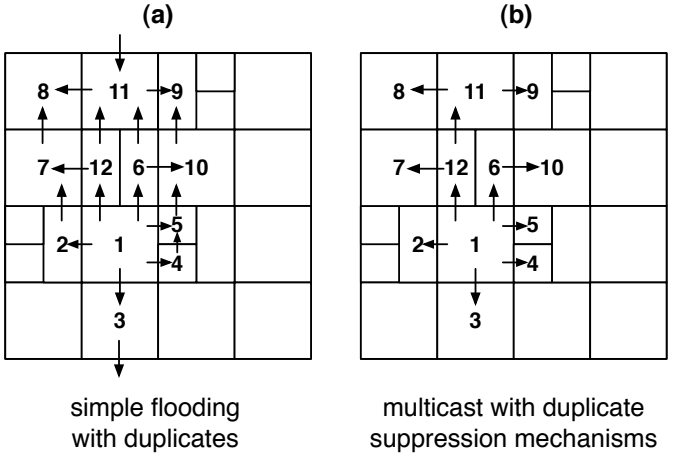


Figure 2.11: Multicasting in CAN

all neighbors, except to the neighbor, from which the message has been received as presented in Fig. 2.11 (a). In order to reduce duplicates, different mechanisms such as duplicate suppression and space distance limits (presented in [136]), are used. Also, the number of dimensions used in CAN influences the forwarding behavior of nodes. By only forwarding messages along certain dimensions, using space distance limits and duplicate suppression mechanisms, duplicates can be reduced as presented in Fig. 2.11 (a). Unfortunately, duplicates can be reduced but not always completely avoided due to the nature of CAN multicasting.

2.5.6 Multicast in Chord

There are two main solutions to support multicast in Chord. The first approach can be easily implemented and uses a forwarder driven approach, where forwarders determine which child nodes they have to serve with multicast data. The second solution is driven by the receivers, which determine and select the parents that have to forward multicast data to them.

Forwarder Driven Multicast in Chord

Multicast in Chord can be done very easily using *forwarder driven multicast*, which works as follows. Each node forwards a received multicast message to all its fingers that are in a certain range. This range is defined by the node from which the multicast message was received.

An example of multicast using Chord is presented in Fig. 2.12. We assume that the root of the multicast tree (node 6) is the node with the lowest ID. The root then sends a multicast message to each finger in its finger table (nodes 8, 12, 23, 39 and 75). A finger specific range information is also sent with each multicast message to the individual fingers. This tells fingers to which other nodes they still have to forward the multicast message. Using the range information, node 39, e.g., has to forward multicast messages only to nodes in the ID range 40 to 74. Therefore, node 39 forwards multicast messages to nodes 45 and 52. The finger specific range is adapted by each forwarding node in order to avoid duplicates.

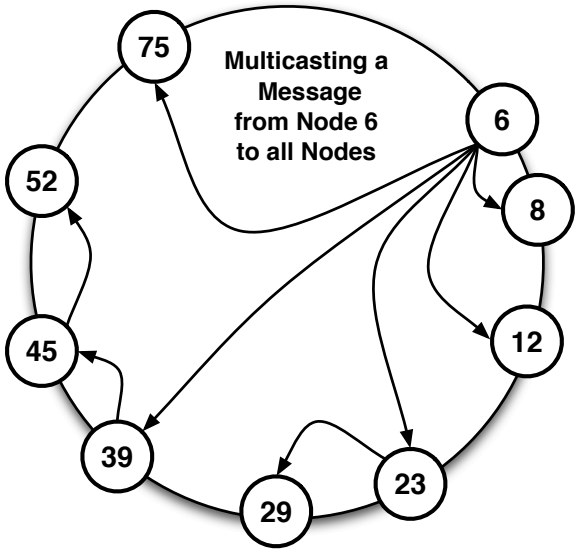


Figure 2.12: Multicasting in Chord

Receiver Driven Multicast in Chord

Using *receiver driven multicast*, nodes select their multicast parent nodes. This is opposed to forwarder driven multicast, where child nodes are determined by their parent nodes. In order to support receiver driven multicast, nodes need to know a few nodes that could act as their multicast parents and then select one of those candidates as their actual parent. This can be supported by introducing backward fingers. A backward finger of a node X points to a node Y, which has X in its finger or successor table.

2.5.7 NICE

Nodes in NICE [15] are arranged in clusters and layers, and are structured hierarchically as shown in Fig. 2.13.

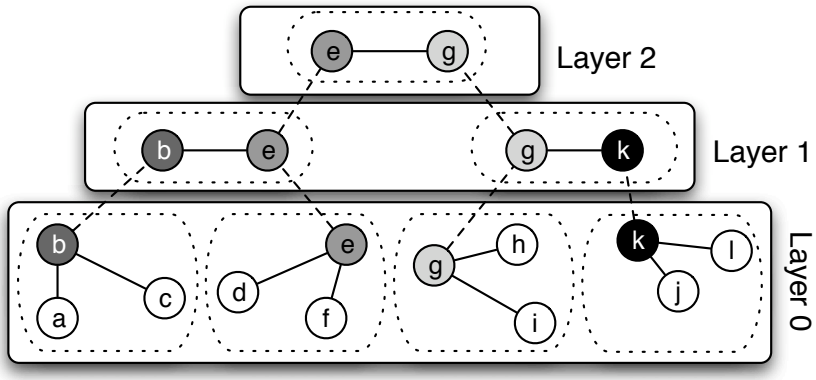


Figure 2.13: NICE Layers and Clusters

Nodes within the same cluster are called cluster mates. Each cluster mate knows its cluster leader and some or all other cluster mates in the same cluster. The size of a cluster varies between the lower bound k and the upper bound $3k-1$, with k being a predefined cluster constant larger than zero. These boundaries help to avoid conflicting maintenance operations, which will be discussed later. One of the cluster mates within a cluster is determined to be the cluster leader. This cluster leader is then a member of a cluster of the next higher layer. Cluster leaders are determined by choosing randomly a node from the so-called graph-theoretic center of clusters. This graph-theoretic center is determined by calculating the eccentricity for each node. The eccentricity is the maximum distance from the node to any other node in the cluster. Nodes with minimum eccentricity then build the center of the cluster. Each layer consists of one or more clusters. They are are ordered from the bottom layer zero to the top layer n. The top layer consists only of a single cluster with one cluster member. This is the root of the NICE network.

The structure of NICE is specified with five invariants. They have to be fulfilled at any time.

1. A node belongs to only a single cluster on each layer.

2. A node located at layer L is also also located at layers $L-1, ..., 0$.

3. A node not present in layer L can not be present in any higher layer ($L+i, i \geq 1$).

4. The size of a cluster is between k and $3k-1$, where k is a constant with $k > 0$.

5. There is a maximum of $\log_k N$ layers and the highest layer only contains one node (the root node).

Cluster leaders multicast data among all their cluster mates in all their clusters. A cluster leader, which would be a member of a cluster in each layer, would have a very high fan out (depending on the cluster size). Note that for n hosts with a maximum cluster size of s, the resulting amount of layers would be $\log_s(n)$. Therefore a cluster leader could have to serve up to $s \log_s(n)$ hosts with the multicast data. The authors of NICE therefore state that it has not been designed for high bandwidth data transmission.

Joining a NICE Network

A node joining a NICE network has to contact first the root node for that specific NICE network. It then receives a list of all cluster leaders on the next lower layer. In the next step, it contacts every reported node and identifies the closest node to itself in terms of round trip time (RTT). This helps to select the most appropriate cluster to join. Therefore, nodes that are physically close are grouped together in a cluster. Afterwards, the joining node sends a new request to the determined closest node. This process is iteratively repeated until layer 0 is reached. There, the new node joins the most appropriate cluster.

Leaving a NICE Network

Nodes can leave a NICE network in two ways, gracefully and ungracefully. In the graceful case, the leaving node announces its departure to its cluster mates before it really disappears. Other nodes can now react to this situation appropriately and perform any handovers from the leaving node that might be necessary. If a node suddenly disappears, then it leaves ungracefully. The cluster leader and the cluster mates do not recognize the departure of the node directly but only by timeouts and missing heartbeat messages. In such cases, parts of the NICE network might not work properly due to invalid states or missing handovers.

Maintenance of a NICE Network

Maintenance of NICE contains several refinement operations to handle the nodes and the tree structure. Heartbeat messages are used to periodically exchange information such as the view of a cluster by a node and its RTT to the other cluster members. The refinement operations might consist of splitting a cluster, merging two clusters, or to determine a new cluster leader. The refinement operations are only invoked by a cluster leader when it detects an invalid or not optimized state in its cluster. Leave and join operations could require the cluster leaders to be changed.

2.5.8 ZigZag

To reduce the high fan-out of NICE presented in Section 2.5.7, an extension of NICE called ZigZag [163] was introduced.

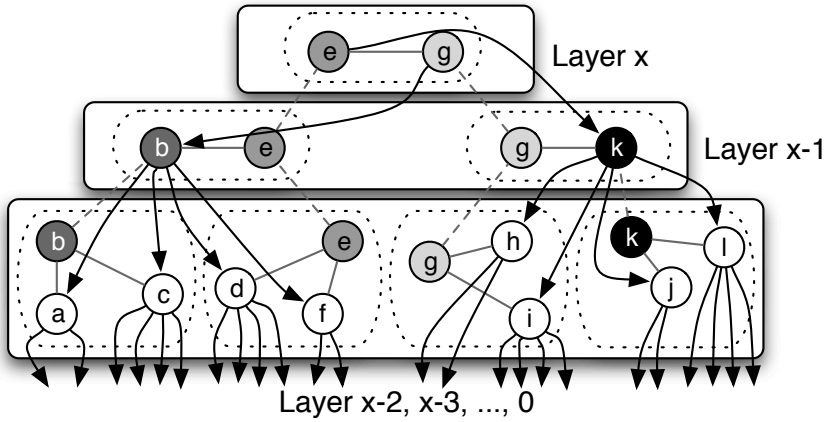


Figure 2.14: Optimizing Fan-Out in NICE using ZigZag

ZigZag distinguishes between the following instances: subordinates, foreign heads, foreign subordinates, and foreign clusters. A subordinate of X is a non-head peer of a cluster headed by peer X. A foreign head of layer $(j-1)$ subordinates is a non-head cluster mate of a peer X at layer $j > 0$. Layer $(j-1)$ subordinates of X are called foreign subordinates of any layer-j cluster mate of X The foreign cluster of any layer-j cluster mate of X is the layer $(j-1)$ cluster of X.

Cluster mates in ZigZag only receive messages from foreign cluster heads, which send multicast data only to foreign subordinates (which are one layer "below" them). A cluster leader *H* for a host *X* has in the next higher layer a cluster mate *M*, which is a foreign cluster head for host *X*. As Fig. 2.14 shows, the fan-out for a cluster head can be reduced using ZigZag. A cluster leader now only has to serve up to $2s-2$ other hosts, but typically only $s-1$ other hosts for a maximum cluster size s. The fan-out of a cluster leader is now independent of the numbers of layers. Worst case fan-out for NICE is $O(k \cdot log_k N)$ and for ZigZag it is $O(k^2)$.

2.5.9 VRing

VRing [159] is an ALM protocol that establishes a virtual ring as an Overlay Network among the multicast group members. This is done in a self-organizing and distributed manner. To reduce the routing delay in the Overlay Network ring, a spare ring overlay structure is formed. This helps to improve the connectivity among group members. The original ring and the spare ring are both used to forward data packets using a data delivery and duplicate suppression mechanism. Each connected component has a leader node. The main communication and organization tasks and building of the Overlay Network are handled by leader nodes.

2.5.10 Borg

Borg [184] is a protocol for scalable ALM dissemination in P2P networks. Borg offers a multicast tree creation methodology, taking both forward-path and reverse-path-forwarding into account. It parts the way from a multicast sender to a receiver into two segments. The segment closer to the receiver will build the multicast tree with reverse-path-forwarding. On the other hand the segment closer to the sender will choose between forward- or reverse-path-forwarding depending on which one of them is better. Borg is built on top of a routing substrate like Pastry described in Section 2.3.3 or Tapestry described in Section 2.3.4.

2.5.11 PeerCast

In [183], an effective passive replication scheme, designed to provide a reliable multicast service, is presented. PeerCast is an efficient and self-configurable ALM framework. Peers in the PeerCast Overlay Network act as clients and servers. PeerCast Middleware is divided into two tiers: "P2P Network Management" and "End System Multicast Management". The authors focus on the development of an analytical model to discuss fault tolerance and to present an effective node clustering technique based on landmarks. This way, PeerCast can cluster end system nodes by using physical network proximity information for fast multicast group subscription and efficient data dissemination.

2.5.12 Dr. Multicast

Dr. Multicast [173] (MCMD) is targeted at data centers. According to the authors, IP Multicast is not widely available in data centers due to various reasons. This includes the fact that IP Multicast is often perceived as a performance degrading technology impacting routing and networking hardware efficiency. Furthermore, the authors claim that IP Multicast applications are often unstable, especially having potential problems when the scale of group members increases. Such multicast storms might even impact normal Internet users, not participating in IP Multicast group transmissions.

MCMD is a system that maps IP Multicast to a combination of unicast as well as IP Multicast transmissions. It relies on unicast connections only in case IP Multicast is not supported in the network. One focus of MCMD is on administrator-specified acceptable-use policies for multicast usage and support in data centers. MCMD tries to take the worries from system administrators in order for them to see IP Multicast not anymore as a threat to their networks. It offers a policy based system to regulate the usage of IP Multicast on per nodes base (allow-join, allow-send, allow IP Multicast, max. sending rate, max. join groups) or even on per network / data center base (max. IP Multicast groups active). In order to enforce these policies, IP Multicast packets are intercepted before being able to leave a node (through the network interface). MCMD then applies the policies and either

sends the packets using IP Multicast or unicast connections using the normal network interface. This also means that applications can use the standard IP Multicast socket interface, but IP Multicast system calls are intercepted. Multicast groups are then translated to a set of unicast and IP Multicast addresses.

The MCMD architecture consists of two components:

- A stateless library module (translation of multicast groups, interception of outgoing multicast messages, and translation of multicast messages to a set of unicast and multicast destinations)
- An MCMD agent (runs as a daemon process on every node, applies the policies and access control lists, and stores the mapping tables for IP Multicast group mappings)

It is important to note that MCMD agents need a designated instance acting as leader. This is required in order to be able to support centrally governed policies and access control lists. The library module and MCMD agent run on Linux only.

2.6 Extensions and Application Frameworks using ALM

2.6.1 Overview

In this Section, we present different frameworks and applications that build ontop and extend ALM networks in order to offer additional overlay services. This includes an architecture for Video on Demand systems [109] presented in Section 2.6.2, Selectcast [24] described in Section 2.6.3, and finally, SplitStream [50] discussed in Section 2.6.4.

2.6.2 Video on Demand using Multicast

As stated in [109], the main bottleneck of Video on Demand (VoD) services is the server's storage I/O and network I/O bandwidth. Using multicast improves the distribution of a video program to multiple clients, hence leading to better performance of a VoD service. The authors critically evaluate and discuss the progress in developing multicast VoD systems. They also present a concept and architecture for multicast VoD and then introduce advanced techniques that can be used in multicast VoD systems. Problems related to multicast VoD services are also analyzed and evaluated.

2.6.3 Selectcast

Selectcast [24] is a self-repairing multicast overlay routing facility running on top of the P2P network infrastructure Astrolab. SelectCast uses a subscribe / publish mechanism to distributed the data. Potential receivers of messages can be defined by using an SQL like syntax on selected attributes of such potential receiving hosts

(up-time, latency, throughput, etc). Astrolab has a DNS-like directory service, in which it organizes its peers. Attributes of leaf domains are readable and writable whereas the attributes of non-leaf domains are aggregated and summarized from its child domains.

2.6.4 Splitstream

Splitstream [50] is an ALM system for high-bandwidth data dissemination. Multicast data is split and then distributed over dedicated multicast trees. These different trees build a so called forest, which is balancing the forwarding load and ensures a certain degree of fault tolerance. Redundant coding of the data can be combined with Splitstream to reconstruct the original data if it gets received at a node only partially. This also helps for cases of node failures and unannounced departures. Nodes are only participating in a limited degree to forwarding the received data in respect to their maximal outbound bandwidth and other constraints. Therefore, Splitstream supports also very well today's most typical asynchronous end user Internet connections like xDSL. The implementation of the authors runs on top of Scribe/Pastry presented in Sections 2.5.3 and 2.3.3. Evaluations have been performed using a simulator and the implementation has also been deployed in the PlanetLab [131, 129, 130] environment. They show that Splitstream is resilient to node failures. Additionally, they also show that the traffic load is evenly distributed over the participating nodes in respect to their bandwidth constraints not overloading them with forwarding traffic.

2.7 Quality of Service

2.7.1 Overview

In the networking scope, Quality of Service (QoS) enables to support different priorities for data flows. QoS requirements or guarantees can be modeled using different parameters, such as:

- delay (time of packet delivery, e.g., in ms)
- jitter (delay variation between packet arrival)
- packet drop probability
- required bandwidth (bit rate)
- error rate (bit errors)

QoS is required if networks are overused and congestion can occur. Especially time critical applications, such as Voice over IP (VoIP) or real-time multimedia streaming (IPTV) profit from QoS guarantees. If networks are over-provisioned and no congestion occurs, then QoS mechanisms might not be needed. On the

other hand, best-effort networks, such as the current deployment of the Internet, do not support QoS out of the box.

2.7.2 Quality of Service in the Internet

QoS mechanisms [25, 22], as well as QoS specifications [179, 155] and characterization [156] for the Internet have been investigated and proposed already over a decade ago. But, QoS support in existing networks is still not widely deployed [57]. Also other aspects of QoS and Denial of Service [153] still need further investigation in order to widely deploy QoS.

DiffServ, IntServ and RSVP

Different approaches on how to support QoS in the Internet exist. Differentiated services (DiffServ) [22] mark packets according to the type of service they need. Routers can then decide how to queue those packets depending on the markings. While DiffServ uses classes to distinguish QoS requirements, integrated services (IntServ) [25] work on flows, which requires more processing overhead. IntServ uses a very fine grained model, wheres a DiffServ is using a more coarsely grained system to perform QoS mechanisms.

To signal QoS requirements, different methods can be used (in-band or out-of-band signaling). RSVP [1] for example is such an out-of-band QoS signaling protocol. To reduce signaling overhead, RSVP requests can be aggregated [13].

EuQoS Project

The goal of the EuQoS (End-to-end QoS support over heterogeneous networks) project [68, 67, 116, 28, 11] is to resolve the required design issues presently associated with the delivery of end to end Quality of Service (QoS) service across heterogeneous networks. It enables end-to-end QoS in heterogeneous systems through the Internet, but it has not been deployed. EuQoS only supports QoS for unicast connections between end-points using DiffServ alike mechanisms as presented in [116]. The task of the Multicast Middleware is to simplify the QoS provision for IP Multicast by mapping multicast communication to unicast.

QoS for Multicasting

Introducing QoS for IP Multicast (e.g., using DiffServ-mechanisms) is challenged with a lot of problems. This is mainly due to the higher complexity of the point-to-multipoint or multipoint-to-multipoint communication that IP Multicast offers. Although the DiffServ architecture is highly scalable, its simplicity causes fundamental problems when trying to be used to enable QoS for IP Multicast. Any node can just send spontaneously and asynchronously IP Multicast data to any IP Multicast group. All paths involved in the distribution then have to be able to provide the required resources in order to support QoS for the individual receivers. Group

membership, and therefore also the corresponding distribution paths are highly dynamic due to nodes joining or leaving the IP Multicast groups. This makes it especially complex to reserve resources in advance for the potential receivers of IP Multicast data. Additionally, the participating group members that want to receive the IP Multicast data can have different QoS requirements. Hence, this has also implications on the distribution trees.

These different problems are also referred to as reservation sub-tree problem, heterogeneous multicast groups, and dynamics of any-source multicast. In [23], these problems are described in more detail and some solutions are presented. In general, when trying to introduce QoS for IP Multicast a lot of problems have to be faced and further considerations regarding scalability, deployment, and security have to be taken into account.

Different approaches exist to cope with the problem of introducing QoS services and reliability to overlay networks [107, 105, 95] and multicast routing [182]. Others cope with introducing QoS to the Internet by the means of using overlay networks [162]. Supporting heterogeneity and congestion control [127] is also a major requirement for a QoS enabled overlay network.

Denial of Service and QoS

QoS and Denial of Service (DoS) go hand in hand since one requirement of QoS should also be to provide protection against DoS attacks [153]. Different mechanisms have to be introduced to make an overlay network robust against fundamental attacks and to ensure reliability in order to support basic QoS requirements.

2.7.3 Quality of Service for P2P/ALM

In order to improve P2P and ALM concepts, Quality of Service (QoS) aspects should be taken into account, which will be one of the goals of this thesis. Different approaches, such as QRON [107, 105], mOverlay [185], HostCast [106] and many others [142, 101, 104, 62, 3, 189] have been introduced to enable different kinds of QoS functionality in P2P/ALM networks.

QRON

QRON [107, 105] uses Overlay Brokers in each access network. It forms an Overlay Service Network for resource allocation and negotiation, overlay routing, topology discovery and other functionality. The goal of QRON is to find an overlay path that satisfies the QoS requests of the involved hosts. But, at the same time, it tries to balance the overlay traffic among the Overlay Brokers and the links of the Overlay Service Network. One disadvantage of this approach is the use of Overlay Brokers, which requires additional infrastructure support in each access network, and therefore is not really an easily deployable solution.

mOverlay

The authors of mOverlay [185] propose a mechanism for constructing an overlay network that takes locality of network hosts into account. Peers in the overlay network are grouped together according their locality. The amount of links between different groups is limited. The overlay constructed using mOverlay can significantly reduce the communication needed between end hosts. The mOverlay approach reduces the average distance between a pair of hosts compared to a randomly connected overlay network. The main focus of mOverlay is on efficiency in terms of communication costs and on scalability. No infrastructure support is needed, but a rendez-vous point has to be deployed. But, mOverlay also has some drawbacks, such as only receiving partially optimal results from the locating procedure and higher/longer resource usage (long run time to determine locality).

Hostcast

Hostcast [106] uses an overlay tree for data delivery and an overlay mesh for control and maintenance. Both of these overlays cover all multicast group members. The data delivery tree is used to disseminate the multicast data. The control mesh is used to transmit control messages and to perform overlay path measurements. Based on the mesh, members can gradually adjust their positions in the data delivery tree to improve their performance. Adjustments performed to the data delivery tree are also applied to the control mesh. HostCast uses a measurement-based approach to determine QoS capabilities of paths. According to these measurement, the data delivery tree is then adjusted to offer the best QoS to multicast group members. Hostcast uses one-way measurements, where probing packets are only sent from the root to the children. This requires that all hosts need to have a synchronized group clock, which is a major shortcoming of the presented approach.

Adaptive Routing Mechanisms for QoS-Constrained Media Streams

An adaptive routing mechanisms for QoS-constrained media streams in overlay networks is presented in [3]. It targets scenarios with scalable overlay topologies to route real-time media streams between publishers and potentially many thousands of subscribers. For a certain number of nodes in a distributed system, the optimal structure to minimize the average distance between any two nodes is calculated. This structure together with a greedy algorithm that selects paths between nodes regarding their real-time improves routing latencies by as much as 40%. This in in comparison to approaches, which do not take physical links costs in terms of delay into account. Nodes are also dynamically repositioned to improve the probability of meeting service requirements for the streaming. The presented approach mainly focuses on improving latency and, up to a certain degree, also link stress, but unfortunately does not seem to take bandwidth requirements of nodes into account.

Case Study of QoS Supporting Group-Conferencing Tool Suite

In [62] a case study of the design and development of a group-conferencing tool suite is described. This suite is built on top of a JAVA-based overlay network event dissemination framework named DEEDS. This framework can be extended via service template plug-ins to support QoS. Such QoS templates deal with two separate streams of events. Multi-point streams that are produced by publish-operations, and unicast streams consisting of feedback events. The paper further describes, how the framework was used to create a simple but effective distributed solution using the appropriate QoS templates. These templates were then evaluated using the built in simulator of DEEDS for 100 backbone nodes during several hours virtual time. Very high packet loss rates of up to 50% were tested to evaluate the QoS performance in extreme situations. The actual group conference suite was only evaluated within a limited dissemination network and with a low number of desktop clients. Hence, the scalability of the presented conferencing suite has not yet been proved.

QoS in Selfish Overlay Networks

Increasing QoS in selfish overlay networks is discussed in [142]. Since overlay networks might have selfish nodes that do not want to contribute but only receive transmitted data, they can have a negative impact on QoS performance. The selfishness ranges from nodes just trying to cooperate enough to stay in the network and still be served up to nodes that are extremely selfish, so called free-riders. These free-riders could even refuse to share any of their resources which has an impact on all other overlay nodes depending on them. The authors propose a game-theoretic approach to analyze and evaluate selfish nodes in order to increase QoS on average for all nodes by detecting and excluding free-riders.

The Impact of Uncooperative Nodes on QoS

A similar problem regarding uncooperative nodes and the impact on QoS is discussed in [104]. A lot of existing overlay networks rely on cooperation of the nodes in order to optimize the overlay topology. Nodes can behave selfish to achieve better QoS or to minimize their forwarding overhead. The authors present a systematic study on the impact of selfishness in both tree and mesh overlay construction. They consider multiple QoS parameters for streaming applications. This includes stream latency, resolution, and continuity. They show that trying to construct a globally optimal overlay is vulnerable to nodes acting selfish. But they give directions on how to design an overlay that is both globally optimal and selfish-resistant.

Live Media Service Grid

A live media service Grid (LMSG) is proposed in [189]. The authors present an extensible middleware architecture for supporting live media applications in a grid

computing environment. The LMSG service uses service brokers nodes. These service brokers nodes have to be strategically deployed in the Internet. LMSG the uses a QoS-satisfied inter-domain overlay multicast algorithm (QIOM). This algorithm organizes service brokers nodes in such way to build a QoS-aware multicast service trees.

Simulations that have been performed show that QIOM helps to provide QoS-enabled overlay services while still balancing the overlay traffic among service brokers nodes. One downside of this approach is the additionally required infrastructure support by having to deploy service broker nodes.

Path-aware Overlay Multicast

Path-aware Overlay Multicast [101] investigates a heuristic overlay multicast approach. This approach is called Topology Aware Grouping (TAG). Using TAG facilitates exploiting underlying network topology data when constructing overlay multicast networks.

TAG can be extended to support delay and bandwidth requirements in order to construct overlays that satisfy certain QoS requirements for end hosts. The authors compare TAG with delay-first and bandwidth-first Narada/End System Multicast (ESM) in a variety of simulation configurations.

In more complex scenarios, TAG can produce multicast trees with a high depth. This increases data forwarding latency which is a major drawback. Also, another shortcoming of TAG is that paths might need to be computed many times if there are many nodes in the same domain. The topology discovery procedure then leads to poor performance.

Evaluation of a Live-Streamed Baseball Game using P2P Multicast

In [4], the authors analyze and evaluate a live-stream P2P multicast distribution of a baseball game. This multicast session was broadcasted during 4 hours with overall 120000 receivers. The video was transmitted at a resolution of 640 times 480 pixels with a rate of 759 Kbps.

At the peak time, 60000 peers received the stream concurrently. Totally, 731 Autonomous Systems (AS) in 51 countries were participating in the P2P network. Around half of the peers received 95% of the media stream correctly. The average duration of stay was just 106 seconds, and it took a peer around 32 seconds before they were able to receive the stream. A high increase of peers joining was noticed towards the end of the baseball game, when everybody wanted to join to see thee most important part of the game. Joining rates had a maximum at 80 peers per second, while the leaving maximum rate was at 328 peers per second.

Unfortunately, the authors do not reveal which P2P content distributer performed the broadcast and also only a one-time event was evaluated. This is a major downside of this evaluation, and makes it hard to see how the gained results can be applied to P2P multicasting of real-time video in general.

2.8 Conclusion

In this Chapter we gave an overview of Peer-to-Peer (P2P) and Application Layer Multicast (ALM) networking. Using P2P/ALM networks helps to overcome certain limitations of the client-server architecture as well as to overcome limited availability of certain services. This is mainly the case for IP Multicasting, which is not widely deployed, and therefore not available to end users. Using ALM allows end users to benefit from the multicast paradigm for efficient data dissemination.

Various approaches on how to support Quality of Service (QoS) for ALM exist. They are though either targeting specific scenarios or protocols and can not be generally used. These approaches are often limited to just improve hop-by-hop latencies or trying to offer redundant paths. Also, selecting the right local group or cluster aims to, e.g., support bandwidth requirements of users. Therefore, these approaches often have limited QoS support only working for, e.g., low bandwidth streaming scenarios or for delay restricted collaboration networks. Existing approaches lack a general end-to-end QoS support for different application scenarios. Rather than matching QoS requests with appropriate QoS guarantees, these approaches are limited to just provide best-effort services to end users.

To enable QoS for various P2P/ALM protocols, we designed the OM-QoS (Quality of Service for Overlay Multicast) framework, which will be presented in more detail in Chapter 3. Furthermore, we built a bridge between IP Multicast and ALM called the “Multicast Middleware”, which will be presented in more detail in Chapter 4. Finally, an efficient Multicast File Transfer Protocol (MCFTP), which offers efficient data dissemination using cooperation, will be presented in Chapter 5.

Chapter 3

Quality of Service for Overlay Multicast

3.1 Introduction

In this Chapter, we present a flexible and general approach to support Quality of Service (QoS) for Application Layer Multicast (ALM). We introduce the Quality of Service for Overlay Multicast (OM-QoS) framework [31, 32, 33, 41, 42, 43, 35, 34, 46, 20, 145, 16], which enables QoS for different Peer-to-Peer (P2P) ALM protocols.

We present and evaluate OM-QoS applied to Scribe/Pastry (see Sections 2.3.3 and 2.5.3), NICE (see Section 2.5.7), CAN (see Section 2.3.5) and Chord (see Section 2.3.6) and show that we can guarantee that all paths in the multicast tree support certain QoS requirements. Using OM-QoS only introduces a slight overhead in terms of delay, hop-count, packet duplicates, and fan-out.

The OM-QoS principles can be applied to many different P2P/ALM networks, either using a protocol dependent approach, where we modify the ALM protocol itself, or using a protocol independent (layered) approach, which does not require strong modifications of the ALM protocols but rather uses multiple inter-connected instances of ALM overlay networks in parallel in order to support QoS.

We first focused on how to make Scribe /Pastry QoS aware. This is achieved by changing Pastry's ID assignment mechanism. This simple modification does not require many changes of the Scribe protocol. We implemented this modification in Freepastry [75] and evaluated it using Freepastry's integrated simulator mode.

Further OM-QoS solutions are evaluated using OMNet++ [124, 166, 167]. OM-QoS applied to CAN serves as the reference implementation for our protocol independent (layered) approach, because there is no obvious protocol specific solution to enable QoS support. Furthermore, we investigated how OM-QoS can be applied to NICE and Chord using the protocol dependent and independent approaches. Finally, using Chord, we implemented a general receiver driven Overlay Multicast approach that also supports certain delay constraints for a node regarding its path to the multicast tree root node in terms of an upper boundary for the

RTT. OM-QoS applied to Scribe was only evaluated using Freepastry's integrated simulator and not using OMNet++.

In Section 3.2, we describe the architecture and design of OM-QoS. The evaluation scenarios for OM-QoS are presented in Section 3.3. Section 3.4 presents the evaluation results. A conclusion is given in Section 3.5.

3.2 Architecture and Design of OM-QoS

3.2.1 Overview

OM-QoS aims to be a general solution for introducing QoS capabilities to different, mainly structured P2P/ALM architectures and is not limited to DHT [97] like systems such as Pastry. There are two approaches. The first one uses modifications of the ALM protocol in order to support QoS. This is called the protocol dependent approach. The the second one using layers to support ALM protocols that cannot be easily modified in order to support QoS. This is called the protocol independent approach or also layered approach.

The protocol dependent approach works for Scribe/Pastry, Bayeux / Tapestry, NICE, and Chord and might also work for other ALM protocols as well. For P2P/ALM protocols that cannot be modified easily in order to support QoS, the layered approach as used for CAN can be applied as a general solution.

We look at scenarios using network provided QoS (hard QoS), where users get QoS guarantees provided by the underlying network infrastructure. This is for example the case for Differentiated Services (DiffServ) or EuQoS [68, 67, 116, 28] mechanisms. Also, we we will look at dynamic environments using measurement based best-effort QoS (soft QoS), where trees have to adapted periodically if QoS fails. So, OM-QoS should be able to support both kind of environments.

3.2.2 QoS for Multicast Trees Design Principles

The OM-QoS framework aims to be a self-managing general solution to incorporate QoS mechanisms into different P2P/ALM protocols. In order to enable QoS for such systems, the multicast distribution tree has to hold certain properties.

As described in [41, 43], a multicast tree has to be built such that the QoS requirements or capabilities are monotonically decreasing on all paths from the multicast root node to the leaf nodes. This the has the following implications to the multicast tree:

- The root of the multicast tree must be the node with the highest QoS requirements.
- Each child node can only have a smaller or equal QoS requirements than its parent node.

To manage different QoS parameters, we introduce the concept of QoS classes, which have a total order (in terms of set theory). This means that for example a QoS class can be a combination of bandwidth, node uptime and CPU requirements. We only consider QoS classes, which have the following properties:

- There is a total order relation for all QoS classes.
- All parameters of the QoS classes are independent of link length and the number of hops in the network.
- The number of QoS classes is finite.

In other words, we require that the QoS classes can be ordered and that they are independent of path length. There are many possible parameters that can be combined into a QoS class, such as bandwidth, maximum packet loss, uptime of a node, hop-by-hop jitter, CPU speed, etc. The nature of the QoS parameters selected and combined depends heavily on the application.

It is just necessary that all the possible QoS classes defined from such parameters must be comparable. Such a QoS aware tree is depicted in Fig. 3.1. The thicker lines represent higher QoS classes, for example just the bandwidth QoS parameter (thicker lines = higher bandwidth requirements) or the hop-by-hop jitter (thicker-lines = less jitter).

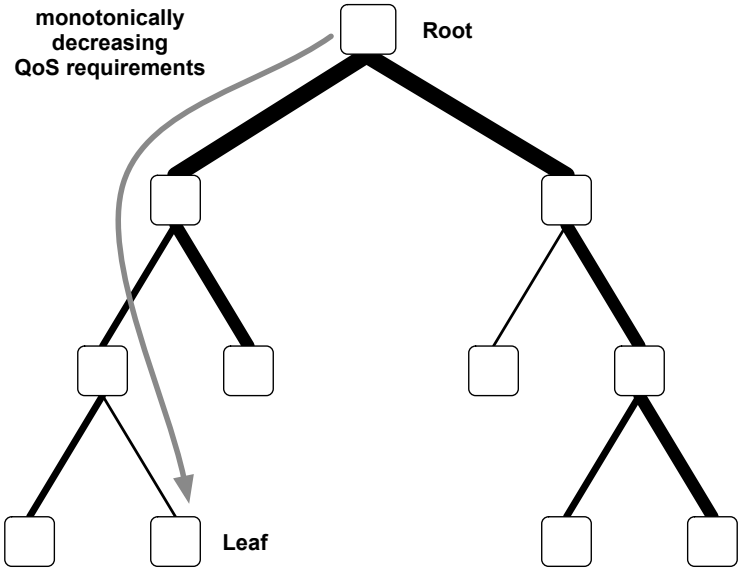


Figure 3.1: QoS Supporting Multicast Tree

Note that in general, there is not always a total order for any possible combination of such parameters. For certain QoS parameter combinations, the order of the QoS classes would have to be defined manually. Such an example of having to manually define QoS classes could occur when combining QoS parameters for

bandwidth, loss and uptime of a node. The different QoS values for the different parameters could be combined as follows:

- Bandwidth: 2.0 Mbps and 2.5 Mbps
- Loss: $<1\%$ and $<2\%$
- Uptime: >1 min. and >15 min.

There are different combinations possibilities when using these QoS parameters. Depending on the application, some parameters have to be prioritized over others. But, this can change even from QoS class to QoS class, and some of the parameters might even be omitted for certain QoS classes. An example of how to combine the QoS parameters into QoS classes could be as follows:

- QoS class 1: bandwidth 2.0 Mbps, loss $<2\%$, uptime >1 min.
- QoS class 2: bandwidth 2.0 Mbps, loss $<1\%$, uptime >15 min.
- QoS class 3: bandwidth 2.5 Mbps, loss $<2\%$, uptime >1 min.
- QoS class 4: bandwidth 2.5 Mbps, loss $<1\%$, uptime >1 min.
- QoS class 5: bandwidth 2.5 Mbps, loss $<2\%$, uptime >15 min.

The example presented above omits certain parameter combinations and also prioritizes the different QoS parameters differently for certain QoS classes (e.g., when comparing QoS classes 3 to 5 with 1 and 2). As already mentioned, this is heavily depending on the application. The same parameters might be combined completely different for another application, where, e.g., loss might have the highest priority.

Also the QoS parameter for maximum delay is not supported by this concept of QoS classes, because parameters that are accumulated over many hops, such as end-to-end delay can not be incorporated into a QoS class. Most P2P networks already minimize the delay between the participating peers. In order to fully support delay-related QoS requirements, the end-to-end delay from the root to a leaf over the in-between hops along the path has to be considered (sum of hop delays). Therefore, we extend the OM-QoS framework to manage end-to-end delay as an additional and from the QoS class construct independent parameter, which will be taken into account when looking for a potential parent in the P2P/ALM network. This allows a peer to request a QoS class (for example a certain bandwidth) for its connection and also ask for an upper bound of the delay (in terms of RTT) it will have to the root of the multicast tree.

The basic idea of OM-QoS is therefore to build QoS-aware multicast trees as presented, with the root of the tree as the sender (core based tree). Therefore, the tree construction mechanisms for the protocol dependent and protocol independent approach of OM-QoS for the investigated P2P/ALM protocols should yield in a multicast tree as shown in Fig. 3.1.

3.2.3 Protocol Dependent Approach

We first look at the concept for providing QoS to multicast in structured P2P networks using a protocol dependent approach. We show on the example of Scribe / Pastry how to enforce QoS aware tree construction in a structured P2P network. We achieve this by modifying the ID assignment method of Pastry based on the QoS requirements of peers. As a result, the multicast tree holds the QoS (for example bandwidth) requirements on each of its end-to-end paths. Then we also show how other protocol dependent modifications enable QoS support for Bayeux / Tapestry, NICE and Chord.

3.2.4 QoS for Scribe/Pastry

If we analyze Scribe's normal multicast tree construction (presented in Section 2.5.3), we can see that the constructed multicast tree does not necessarily hold the properties of a QoS aware multicast tree as described in Section 3.2.2. The reason for this is that end-to-end paths from leaves to the root are more or less randomly chosen, due to random positioning of Pastry (see Section 2.3.3) peers.

Our evaluation in Section 3.4.2 shows that when using Pastry's default peer ID assignment, less than 40% of all end-to-end paths hold the previously described property of QoS aware trees for randomly assigned QoS classes to peers. Because Pastry's default ID assignment does not take QoS requirements of peers into account, multicast trees constructed by Scribe are only by chance holding this property. If only one link in an end-to-end path does not hold the property, holding QoS guarantees for all nodes in the multicast tree below this link is disabled.

To enable QoS for Scribe/Pastry and to enforce the creation of QoS aware multicast trees as described in Section 3.2.2, the following modifications of Pastry's ID assignment method have to be performed:

- For each multicast group exists a dedicated Pastry P2P network. The reason for this is to have only peers interested in receiving the multicast data as potential forwarders. In a normal Scribe/Pastry environment, peers will also have to forward traffic for topics, to which they are not really interested in.
- In this Pastry network exists only one topic. This topic's ID is the highest possible topic ID.
- Since the the QoS requirements of a peer can be higher than its QoS capabilities, we choose the QoS class, which corresponds to the minimum of both, which then is the QoS class for this peer.
- Higher QoS Classes of a peer result in a higher Pastry ID. Therefore, the peer's ID is directly proportional to its QoS class value.
- All peers subscribe to the highest possible topic ID. The peer with the highest QoS class will become the root for this topic / multicast group.

- The ID space is partitioned into segments (see Fig. 3.2): one segment for each QoS class. The order of segments depends on the order of the QoS classes. The lowest QoS class is located in the lowest segment and the highest QoS class is located in the highest segment. As previously mentioned, the assignment of IDs to joining peers depends on their QoS class. The peer ID is randomly chosen within the corresponding segment of the ID space for the peer's QoS class.

There are different possibilities on how large the segments should be, they do not necessarily have to be all of the same size and can for example decrease in size towards the root's ID (highest ID possible). The partitioning strategy has an impact on the construction of the multicast trees and therefore on how well and evenly the overall traffic load will be distributed among the participating peers.

These design choices ensure that all the paths from the root to the leaves will have monotonically decreasing QoS requirements. As shown in Section 2.5.3, the routing path from a peer with a lower ID to a peer with a higher ID always contains peers with increasing IDs. Because the root node of the multicast tree has the highest possible Pastry ID, we enforce that the routing always uses peers with increasing Pastry IDs for the hops on its path from leaf nodes towards the root node. By assigning peer IDs proportional to the peer's QoS class, we enforce a construction of Scribe multicast trees, which hold the decreasing QoS requirement property for each end-to-end path from the root to the leaves. For each node that is passed through on the path from the root node to a leaf node, the QoS requirement of the intermediate node is the same or lower than of its parent node.

Figure 3.2 shows a Pastry ring with a partitioned space. In this example there are three different QoS classes, which are not distributed evenly among the ID space: low QoS, medium QoS, high QoS. The QoS class segments have to be assigned in such a way that for each higher QoS class at least one additional digit (from left to right) matches the topic's ID compared to the previous lower QoS class. Depending on their QoS class, peers get an ID from the respective partition of the ID space. The root node will have the highest possible Pastry ID and therefore will also be in the partition for the "high QoS" class. When a peer now subscribes to a Scribe topic, the join message is routed towards the root node (having the highest ID) and always passes nodes in the same or higher QoS class partition, and therefore on each hop passes through a peer with a higher ID than itself.

When a Scribe multicast message is now disseminated in this Pastry ring, the reverse paths for sending the join messages in Fig. 3.2 are used. Then, all paths from the root node **FFF** to the leaves will on each hop either pass through hops with the same or lower QoS requirements than on the previous hops. Therefore, the previously described property for monotonically decreasing QoS requirements in the multicast tree is fulfilled.

We will show in the evaluation in Section 3.4.2 that basic properties such as the average end-to-end path length of Scribe/Pastry are not changed significantly by introducing the previously mentioned modifications of the Pastry ID assign-

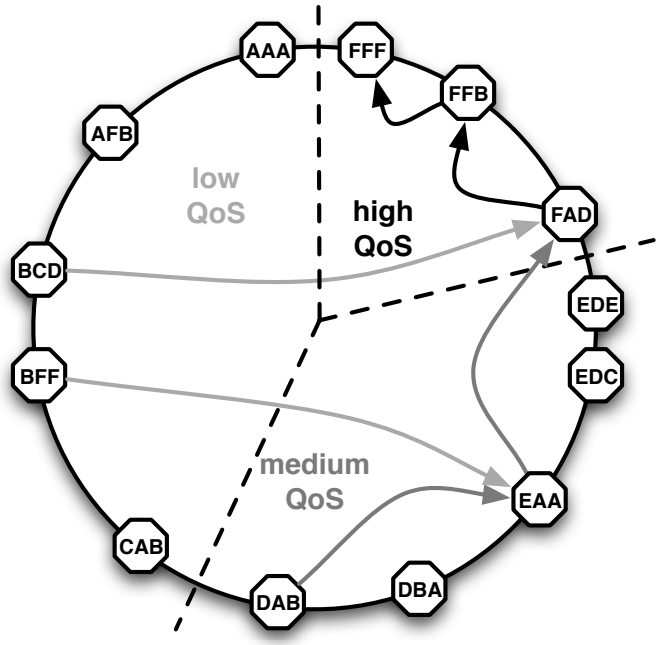


Figure 3.2: QoS Aware Distribution of Peer IDs for Pastry

ment, even though the ID distribution does not follow anymore strictly a normal distribution.

3.2.5 QoS for Bayeux

To enable QoS aware multicast tree setup with Bayeux (presented in Section 2.5.4), a similar mechanism can be used as with Scribe. Due to the different prefix matching order and the non-reverse setup (using *tree* messages instead) of the paths as has been described in Section 2.5.4, the approach has to be slightly modified compared to Scribe/Pastry as follows: to ensure that all paths from the root to any group member hold the basic property described in Section 3.2.2, higher QoS requirements of a peer results in more digits matching the root's ID (right to left). Also, one dedicated P2P network is used per multicast group. Therefore, nodes only forward multicast data that they are actually interested in. This would not be the case for a shared P2P network for multiple multicast groups, where nodes would have to forward traffic for multicast groups to which they are not actually subscribed to.

3.2.6 QoS for NICE

To introduce QoS to NICE (presented in Section 2.5.7), we had to take a completely different approach than for Scribe/Pastry or Bayeux, but at the end only the cluster leader determining mechanism has to be modified: the cluster leader is determined

by the highest QoS class inside a cluster (and not the graph-theoretic center using RTT as metric). An example is shown in Fig. 3.3. Clusters are still built using RTT

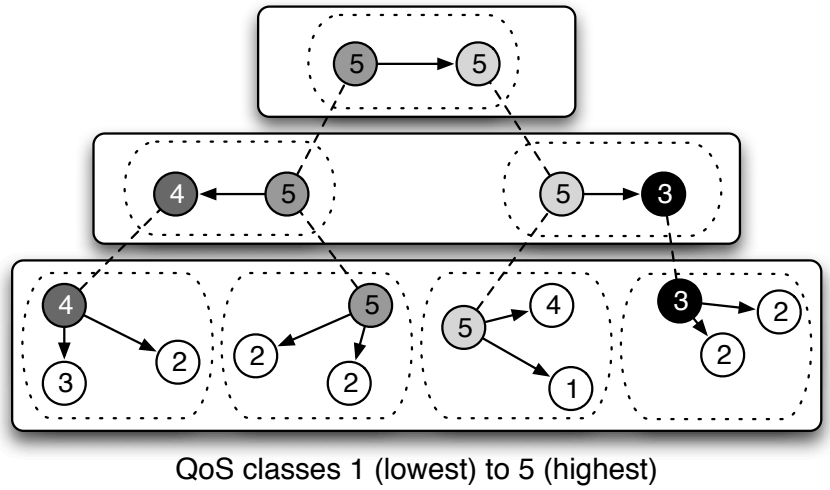


Figure 3.3: QoS Support for NICE with Cluster Leaders having Highest QoS Class

measurements, and we have five QoS classes between 1 and 5, with QoS class 5 as the highest one. Cluster leaders are now hosts having the highest QoS class in the cluster.

The problem of NICE is the high fan-out of nodes acting as cluster leaders in multiple layers. ZigZag (see Section 2.5.8) reduces the impact of the problem with the high fan-out in NICE. Unfortunately, the presented approach to enable QoS for NICE does not work with ZigZag. Figure 3.4 shows that the proposed modification of NICE does not work with ZigZag, because foreign cluster heads can have lower QoS classes than some of the hosts in the cluster to which they act as foreign head.

The *ZigZag problem sub-path* presented in Figure 3.4 shows that the path from the root to the foreign cluster head in the next lower layer still holds the QoS path property of monotonically decreasing QoS classes (path built from QoS class 5 to QoS class 3). But, from this cluster leader to the next foreign cluster head in the next lower layer, the property does not hold anymore (path built from QoS class 3 to QoS class 4). This is due to the fact that there is no correlation between the cluster leader determination in a cluster and its foreign cluster's cluster leader determination. In order to still reduce the fan-out without having to use ZigZag, a delegation mechanism can be used.

To reduce the fan-out with QoS support, the delegation mechanism (adding an additional hop) can be used as follows: cluster leaders delegate message dissemination for each of their clusters to cluster mates with equal or next lower QoS class as themselves. This results in a fan-out up to $\log_s(n)$ for a cluster leader (s = max. cluster size, n = amount of hosts). One drawback of the delegate mechanism is though that an additional hop is introduced. Figure 3.5 shows an example with the delegation mechanism.

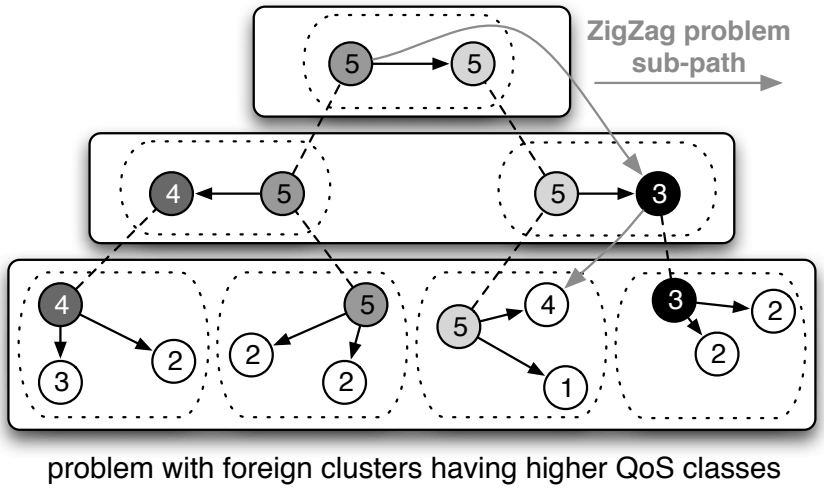


Figure 3.4: Problem why QoS does not Work for ZigZag

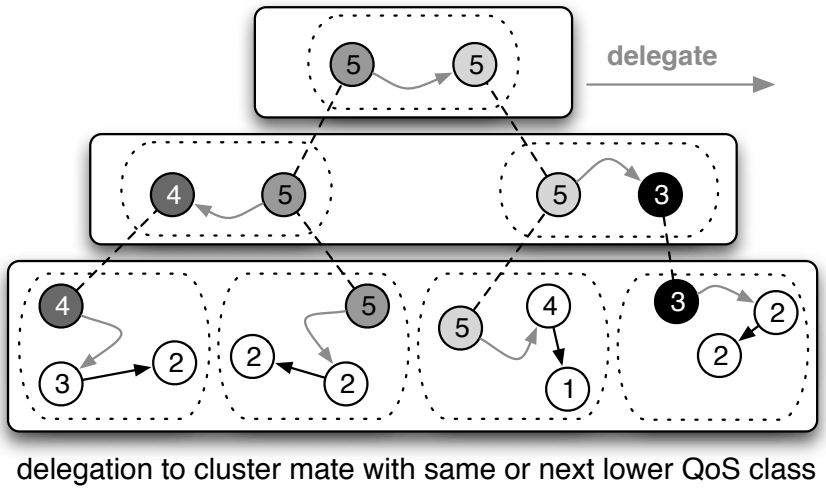


Figure 3.5: Reducing Fan-Out with a Delegate and QoS Enabled NICE

Supporting End-to-End Delay Constraints in NICE

To support also end-to-end delay constraints, we introduce end-to-end delay as an additional parameter. This is independent from the QoS class construct and allows a peer to request a QoS class (for example a certain bandwidth) for its connection and also ask for an upper bound of the round trip time (RTT) to the root of the multicast tree. We call this upper bound *node to root RTT constraints*.

We only take node to root RTT constraints into account when a node joins the NICE network. When a node joins a NICE network, it automatically participates in a multicast distribution tree for that NICE network, and starts receiving and possibly also forwarding multicast data. We implemented this mechanism to support

node to root RTT constraints as follows. When a newly joining node measures the RTT to the potential cluster leaders on each layer, it also asks the potential cluster leaders on all layers for their current node to root RTT value. The cluster leader that fulfills the node to root RTT constraint as closely as possible is then selected for the next iteration. That means the node offering the maximum node to root RTT that is still below the node to root RTT constraints (with a certain safety margin) is selected for the next step in the next lower layer. This is continued until we reach layer 0, where the node joins a cluster that fulfills the node to root RTT constraints.

As an example, we can assume that we have a NICE network with 4 layers. In the top layer 3, only the root is present in its own cluster. A newly joining node asks the cluster leader in layer 3 (the root node) for its cluster mates in layer 2. These cluster mates are cluster leaders of clusters in layer 1 and also at layer 0. The joining node then measures the RTT to these reported nodes and asks them for their node to root RTT. Now, the joining node can determine its own node to root RTT if it would join a cluster of the previously reported nodes. Out of these previously reported nodes, it selects the node that offers the maximum node to root RTT below its own node to root RTT constraint. It then contacts the selected node to ask for that nodes' cluster mates in layer 1. These reported nodes are cluster leaders of clusters in layer 0. Then, determining the node to root RTT matching the node to root RTT constraints are performed again as described before. Finally, the node joins the cluster in the layer that offers the maximum node to root RTT just below its own node to root RTT constraints.

3.2.7 QoS for Chord Multicasting

Forwarder Driven Multicast for Chord

Only a simple modification of Chord (presented in Section 2.3.6) is necessary to enable the previously mentioned creation of QoS aware trees. Nodes in Chord will have to be ordered by their QoS classes. This means that on the Chord ring, we will have clockwise monotonically decreasing QoS classes. The higher the ID of a node the lower is the QoS class of that node (low QoS class = low QoS). We use a core based tree with the root of the multicast tree as the node with the smallest Chord ID.

All multicast messages will then be routed using the forwarder driven multicast approach. A multicast message will always be forwarded in clockwise direction on the Chord ring. Therefore, it will always be sent to a node having a higher ID, hence having the same or a lower QoS class.

An example with three QoS classes is shown in Fig. 3.6. The Chord ID space is split into three partitions. The partition holding the lowest IDs of Chord is reserved for nodes that require high QoS. The next partition holds nodes with medium QoS requirements. Finally, nodes that require low QoS are in the partition holding the highest part of Chord IDs. Since the multicast root is the node with the lowest Chord ID, multicast data dissemination holds the QoS path properties described

before. All paths from the root to the leaf nodes have monotonically decreasing QoS requirements.

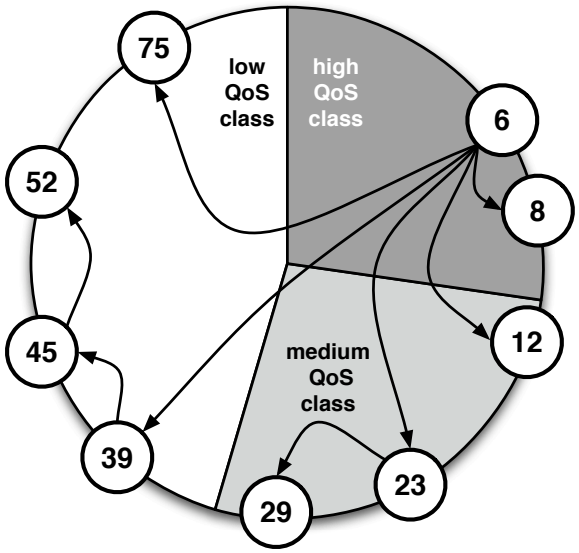


Figure 3.6: QoS Support for Chord Multicasting

Chord Improvements

In order to support QoS using Chord, we had to make some adjustments and enhancements of the original Chord protocol.

By default, Chord is a well functioning and structured P2P network that is highly scalable. But, when fast changes occur in a Chord network (multiple leaves and joins for example), multicasting reliability becomes a problem. Since some nodes might have not yet updated their finger tables and successor lists, some nodes might not be served with multicast messages. Generally, big gaps could occur in the multicast distribution. Therefore, we made some optimizations to ensure more robustness and reliability as follows.

- **Improved stabilization:** Modification of pointers (fingers or successors) causes execution of stabilization directly instead of only periodically. Therefore, errors in the finger and successor tables (invalid pointers) are corrected earlier.

- **Predecessor self discovery:** A node is able to search and find its predecessor by itself. Therefore, stabilization can now rely on a node having up-to-date and correct predecessor information.

- **Self error correction:** A node can add one of its fingers as successor if it does not find a successor. This repairs a broken ring.

- **Improved finger table:** Fingers can change their position in the finger table. Fingers are only deleted if their IDs exceed the nodes' current Chord ID + $2^{(fingerindex)}$.

- **No duplicate fingers:** More distinct fingers facilitates having a more balanced distribution of the multicast tree.

- **Backward fingers:** Each node maintains a list of nodes, that have an entry in their finger table pointing to it. This helps to eliminate multicast errors.

- **Multicast optimizations:** We limited the multicast fan-out (number of nodes that a multicast parent has to serve with multicast data) of nodes to have an upper bound for the maximum fan-out and to avoid that too many nodes directly connect to the root when end-to-end delay QoS requirements have to be fulfilled. Furthermore, we added additional mechanisms to improve multicast robustness.

- **Improved Multicast routing:** Because of the multicast fan-out limitations, most nodes will have to select receivers from their finger list. Therefore, the tree is better distributed and the hop count can be decreased.

We required those optimizations in order to support QoS by having a more robust/reliable Chord version.

Receiver Driven Multicast for Chord

Besides offering QoS using the class construct as described in Section 3.2.2, we also wanted to support guarantees for the RTT between a multicast receiving node and the multicast root node. Nodes can have a certain constraint regarding this so called *node to root RTT* for multicast messages. Therefore, they would only connect to a parent node that would support those constraints. When using the forwarder driven multicast approach, a node is not able to select its own parent for multicast delivery that would match its node to root RTT constraint. Therefore, supporting node to root RTT only works with the receiver driven multicast approach, where children can explicitly chose their multicast parents.

To reduce the overall multicast fan-out of nodes in general as described in Chapter 3.2.7, we limited the maximum fan-out of a node to 7. This also helps to avoid that all nodes try to select the root as their parent, which is often the best candidate to fulfill node to root RTT constraints of nodes. Therefore, a potential parent can reject a child's request if it exceeds a certain number of children. As a consequence, it may take some time to find a parent that satisfies the node to root RTT and that can still accept new children. During this time, a node is not able to receive multicast messages.

3.2.8 Protocol Independent Approach

In the previous sections, we looked at possibilities on how to modify the internal mechanisms of P2P/ALM protocols in order to support QoS using protocol dependent OM-QoS mechanisms. However, such approaches are not always possible, either there are no mechanisms or design changes that could enable protocol-dependent QoS mechanisms or there is no easy way to modify the code or protocol.

Problems to Support QoS for CAN

As an example we will show how the first idea of an approach to enable QoS for CAN (presented in Section 2.3.5) will fail.

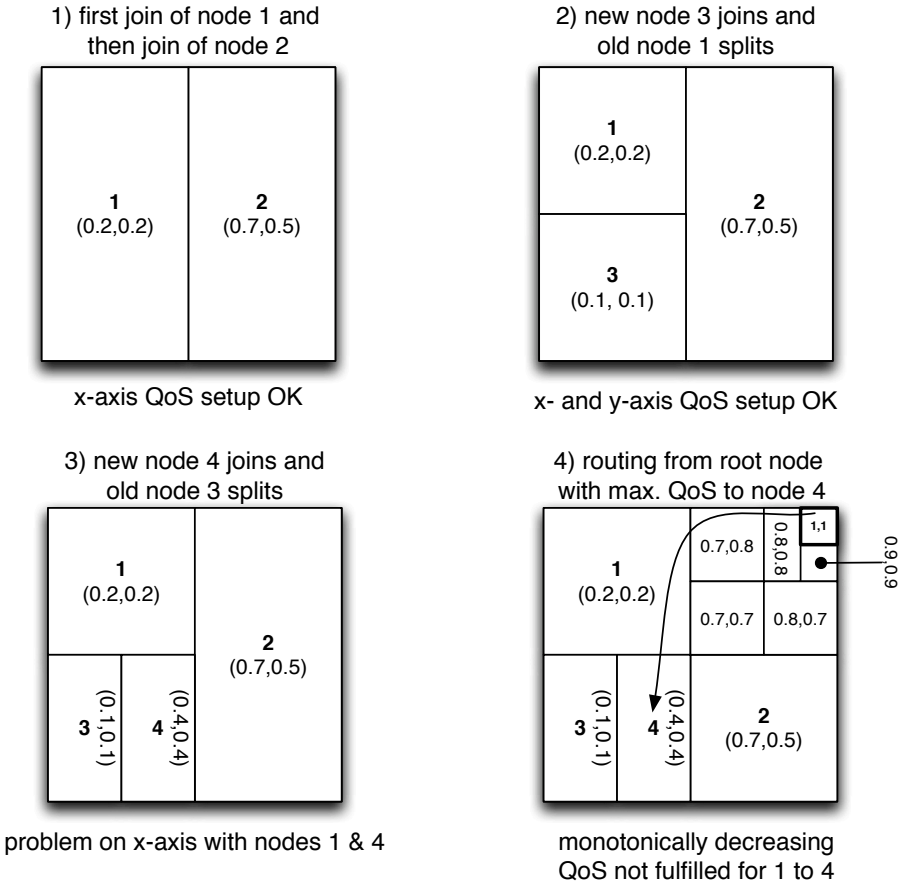


Figure 3.7: Problems with QoS for CAN using Coordinates for QoS Classes

Introducing QoS to CAN by mapping QoS classes to initial coordinates does not work, because the location (zones) of a node can change overtime (zones that are assigned to nodes move over time). Figure 3.7 shows an example where this approach fails. The idea is that higher QoS classes result in higher initial coordinates, so the QoS classes are directly proportional to the initial coordinates of the CAN

node. The problem is that splitting zones can lead to have hosts with higher QoS (initial coordinates) being positioned "below" hosts with lower QoS coordinates. In the example depicted in Fig. 3.7, node 4 (with QoS 0.4,0.4) is placed "below" node 1 (with QoS 0.2,0.2). The property described in Section 3.2.2 does not hold: the QoS requirements are not monotonically decreasing for a message being routed from the root (highest QoS = highest coordinates) to node 4.

QoS for CAN using a Protocol Independent Approach with Layers

To enable QoS for CAN a different approach has to be chosen. This can be achieved using a layered approach. An example with four QoS classes is shown in Figure 3.8.

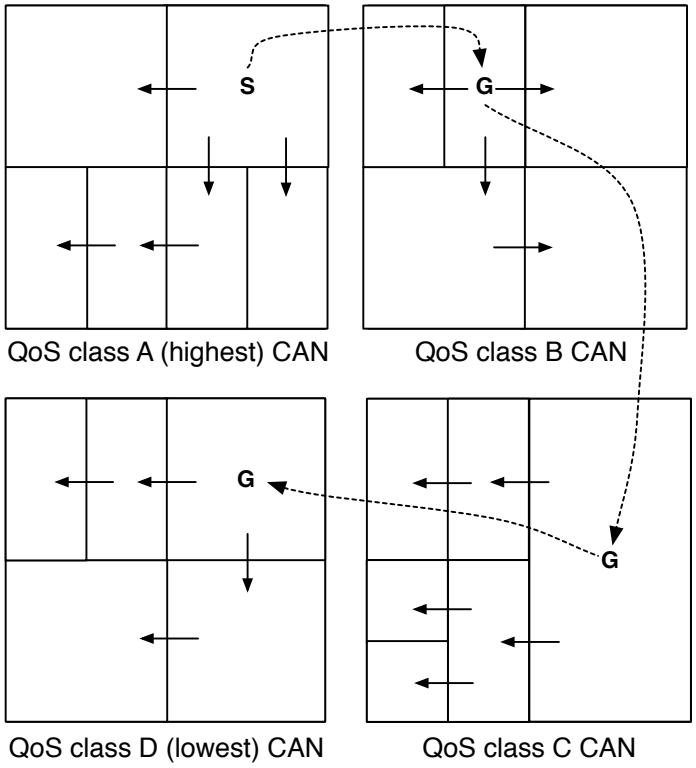


Figure 3.8: QoS for CAN with Layers

Each QoS class has a dedicated CAN (one layer). Hosts having the same QoS class join the same CAN (same layer). The different layers are interconnected using gateway nodes, connecting adjacent layers (in terms of QoS) with each other. The sender **S** disseminates the multicast data inside its own CAN and then sends the data to gateway nodes **G** in the CAN with the next lower and next higher QoS class. The gateway nodes receiving multicast messages from another CAN disseminate them in their own CAN and send them to the CAN with next higher or lower

QoS class depending on the initial "direction" (up or down). This approach guarantees that the resulting multicast tree in such a layered group of CAN networks will follow the described property of monotonically decreasing QoS requirements (from root to leaves). Using layers is the basic idea of the general, but protocol independent OM-QoS approach works, where for each active QoS class a dedicated P2P/ALM instance will be created.

Optimizing Hop-Count and Latency for the Layered Approach

To reduce the overall hop-count and latency from the root to a leaf node, the different QoS layers can not only be interconnected by single adjacent steps, but links can be established jumping over multiple layers as shown in in Fig. 3.9. Normally a layer would only forward the message to its next adjacent layer, for example layer n would forward the message to layer $n+1$ and then layer $n+1$ would forward the message to layer $n+2$, and so on. To optimize this behavior, layer $n+1$ could also forward the message to layer $n+4$, therefore reducing the hop count needed to reach, e.g., layer $n+5$. If a layer would receive a duplicate multicast message (for example layer $n+5$ would receive the same message from layer $n+1$ and layer $n+4$), it would only forward the message once to avoid further duplicates. This mechanism helps to improve the reliability and latency of inter-layer forwarding, but introduces some additional duplicates.

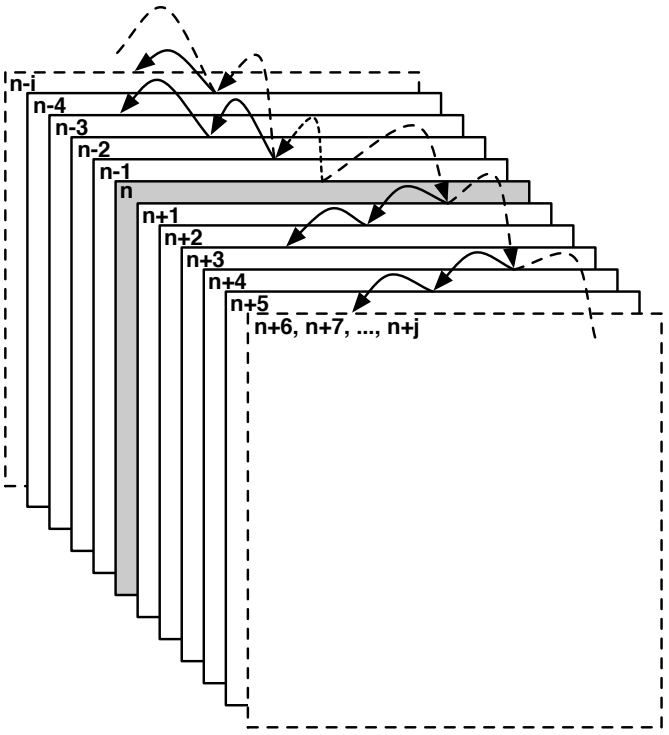


Figure 3.9: Optimizing Layer Hop-Count

To further optimize the reliability, we introduced backup-links between the layers as presented in [27]. If a gateway node fails, the adjacent gateway nodes from the lower or upper layer already have backup links to alternative gateway nodes ready to takeover the inter-layer transfer of the multicast messages from the failing node on the fly.

3.3 Evaluation Scenarios for OM-QoS

3.3.1 Overview

In the following Sections, we present the simulation scenarios used to evaluate OM-QoS applied to different P2P/ALM protocols. First, the scenarios and simulation environment for Scribe/Pastry using Freepastry's built in simulator are presented in Section 3.3.2. Section 3.3.3 presented the common scenarios and parameters for the evaluation using the OMNet++ network simulator. The protocol specific scenarios and values that were evaluated are presented in Section 3.3.4 for CAN, Section 3.3.5 for NICE, and finally in Section 3.3.6 for Chord.

3.3.2 Scribe/Pastry Evaluation Scenarios

In order to prove that our OM-QoS approach works, we performed our first evaluations as a proof-of-concept using Freepastry's built in simulator. The simulations were performed using a simulation environment with limited resources (regarding CPU, memory, availability and number of PCs). For our evaluation we generated network topologies using the BRITE [112, 30] network topology generator. We varied the number of hosts between 100 and 800 in steps of 100. For each number of hosts we generated 40 topologies. We used only a limited number of hosts and topologies due to the limited resources available for performing the simulations. Each topology was generated using Waxman's model [177] with the parameters presented in Table 3.1.

Table 3.1: Parameters used for BRITE

Router model	Waxman
α	0.15
β	0.2
Main plane	5000 squares
Inner planes	5000 squares
Node placements	random
Growth type	incremental
# of neighboring nodes	2

Each such network topology is converted to a distance matrix by finding the optimal route (optimizing the number of hops) for each host pair and calculating the round trip time for that route. Additionally, we assigned to each host a random QoS class (as a value between 0 and 255).

The generated distance matrix and the QoS classes are then used to construct a multicast tree using Scribe/Pastry with our proposed ID assignment method and with the default (random) ID assignment method. The construction of the multicast tree is performed using the "BasicNetworkSimulator" provided by Freepastry [75].

We evaluated the properties of multicast trees created by Scribe/Pastry using our modification of the ID assignment of Pastry as described in Section 3.2.4 by comparing it to the default (random) ID assignment of Pastry. We use hard QoS scenarios and therefore assume that the underlying network can provide the required QoS by performing the corresponding QoS reservations as mentioned in Section 3.2.1.

For each generated multicast tree we evaluated whether the QoS property holds as described in Section 3.2.2 for all end-to-end paths from the root to each node. If at least one hop on the path does not hold this property, we would not be able to guarantee the QoS requirement for that peer and all other nodes below this one in the multicast tree.

When creating Pastry nodes with the modified ID assignment method, this value is used to generate the 32 first digits of the Pastry ID. This way we partition the Pastry ID space into 256 segments of equal size. Nodes with the same QoS class get placed in the same segment. Therefore, a lower QoS class reflects in a lower Pastry ID. The lower the Pastry ID is, the further away a node is in terms of numerical ID proximity from the multicast root node, which gets the highest possible Pastry ID assigned.

For each end-to-end path we check if the following property holds: each step on the path towards the root node has to go through a node with higher or same QoS class value than its preceding node. To verify our proposal, we compare for how many percent of the end-to-end paths the QoS property holds for the random and our ID assignment method. To see whether our proposal produces longer end-to-end paths, we also compare the average and maximum path length for both ID assignment methods. In terms of implementation, we created a new ID assignment factory using the OM-QoS method, which takes the QoS class of a peer into account when determining the Pastry ID.

3.3.3 Common CAN / NICE / Chord Evaluation Scenarios

For the evaluations of CAN, NICE and Chord using OMNet++ [124, 166, 167], we use a simple P2P message based model. We did not use any additional OMNet++ libraries or frameworks, just a basic OMNet++ version 3.3 installation.

The latencies between all the nodes are determined using distance matrices, defining the latencies for each possible node pair connection. The distance matrices were built using topologies generated by BRITE [112, 30] using the same

parameters as for Scribe/Pastry evaluation scenarios presented in Table 3.1 in Section 3.3.2. The minimum, maximum and median delay values of the matrices used for the OMNet++ evaluation scenarios (CAN / NICE / Chord) are depicted in Table 3.2.

Table 3.2: Delay Properties of Distance Matrices in ms

Matrix	**min RTT** (ms)	**mean RTT** (ms)	**max RTT** (ms)
Matrix 0	0.08	22.47	48.44
Matrix 1	0.09	30.35	90.48
Matrix 2	0.05	30.56	94.58
Matrix 3	0.05	29.76	90.23
Matrix 4	0.07	23.26	57.52
Matrix 5	0.09	22.78	51.82
Matrix 6	0.04	22.77	49.24
Matrix 7	0.08	23.27	52.30
Matrix 8	0.05	22.91	53.92
Matrix 9	0.05	23.27	50.83
Matrix 10	0.08	22.47	48.44
Matrix 11	0.05	22.91	54.00
Matrix 12	0.01	23.13	54.17

For each of the scenarios, we looked at various networks ranging with a node count from 100 to 2000 in steps of 100. Each node step was evaluated using 13 different distance matrices, which we evaluated each with different random seeds influencing arrival, departure and other random based decisions.

We use the seeds as input for the random number generators (RNG). For our implementation, we used the Mersenne Twister RNG [111], which has a period of $2^{19937} - 1$ and assures the 623-dimensional equidistribution property.

The seeds used for the simulations (CAN / NICE / Chord) are depicted in Table 3.3. We either used all 20 random seeds (for CAN) or the first 3 seeds from the list (for NICE and Chord).

To evaluate the different scenarios for CAN, Chord and NICE, we analyzed and compared different values. The common values evaluated for CAN, NICE and Chord are multicast fan-out, multicast hop-count, percentage of multicast received, node to root QoS and node to root RTT. The meaning of these values is presented in Table 3.4.

3.3.4 CAN Evaluation Scenarios

We evaluated the application of OM-QoS to CAN using the OMNet++ simulator as described in Section 3.3.3. Therefore, we implemented the CAN protocol and made some enhancements to fully support join and leave functionality.

Table 3.3: Random Seeds used for the Evaluations

Seed #	Value	Seed #	Value
Seed 1	1768507984	Seed 11	640572720
Seed 2	33648008	Seed 12	1569615780
Seed 3	1082809519	Seed 13	1142429693
Seed 4	703931312	Seed 14	307193866
Seed 5	1856610745	Seed 15	34708029
Seed 6	784675296	Seed 16	97450298
Seed 7	426676692	Seed 17	743126457
Seed 8	1100642647	Seed 18	593716555
Seed 9	1359921031	Seed 19	910097052
Seed 10	1209575029	Seed 20	449294716

Table 3.4: Common Values Evaluated in Simulation Scenarios for CAN, NICE and Chord

Multicast fan-out	describes the number of neighbor nodes a node has to serve with multicast data.
Multicast hop count	depicts the number of hops required to reach the root of the multicast tree.
Percentage of multicast messages received	describes the percentage of multicast messages received per node.
Node to root RTT	denotes the round trip time (RTT) from a node to the root of the multicast tree.
Node to root QoS	shows the percentage of paths, which fulfill the QoS requirements of nodes.

We use hard QoS scenarios and therefore assume that the underlying network can provide the required QoS by performing the corresponding QoS reservations as mentioned in Section 3.2.1.

Due to the possibility of nodes leaving at any time during the simulation, we also needed to support temporary key-space assignment as well as zone handover and merge mechanisms. If the multicast sender leaves the CAN network, its responsibility to transmit continuously multicast messages has also to be passed to another CAN node, which then becomes the new multicast root. The CAN protocol was implemented in a fully decentralized manner, the network was built and maintained using only message transmission among nodes. Therefore, it would be quite simple to modify our solution to also work in real networks.

We analyzed the CAN implementation and our OM-QoS extensions to CAN

using various scenarios. We present the results for a scenario consisting of a CAN network without QoS support, where the nodes have one of 64 possible QoS classes assigned, in order to verify the QoS properties of the multicast paths for native CAN. Further, we present the results for a scenarios consisting of an OM-QoS enabled CAN with one of 64 QoS classes assigned to a node. We also discuss the results for a scenario with 256 QoS classes in an OM-QoS enabled CAN. All nodes get their QoS class (out of 64 or 256 possible QoS classes, depending on the scenario) assigned on startup.

We evaluated all values as shown in Table 3.4 in Section 3.3.3 and additionally the CAN specific values as presented in Table 3.5.

Table 3.5: Additional Values Evaluated in Simulation Scenarios for CAN

Average multicast duplicates per multicast message	denotes the average number of duplicated multicast messages received by a node
Time until a node has joined	is the time a node takes to join the CAN network, this includes finding the node responsible to split the zone.
Time until a node has left	is the time a node takes to gracefully leave the CAN network, this includes zone handover and synchronization for overlapping leaves of neighbor nodes.

The latencies between all the nodes are determined using distance matrices, defining the latencies for each possible node pair connection as described in Section 3.3.3. We used for each scenario all 20 random seeds presented in Table 3.3 in Section 3.3.3. The random seeds influence different random based decisions, such as the QoS Class a node has, the departure time of a node, failing QoS requirements, etc. Therefore, each scenario consists of 5200 simulations. We removed 1% of the outliers (0.5% of the minimum and maximum values each) for the figures in the CAN evaluation results presented in Section 3.4.3.

3.3.5 NICE Evaluation Scenarios

Our evaluation of QoS for NICE has been performed using the OMNet++ simulator as described in Section 3.3.3. We implemented the basic NICE protocol and made some enhancements to support further reliability, such as handshakes for cluster-leader transfers, split and merge operations, leaving and root transfers. Our implementation is fully decentralized, the overlay structure was setup and maintained using only message transmission among the NICE nodes.

The QoS aware NICE has been evaluated using hard QoS guarantees offered by the underlying network and soft QoS as described in Section 3.2.1. Hard QoS

could be achieved using, e.g., DiffServ or EuQoS [28] mechanisms. Soft QoS is a measurement based best-effort QoS. Nodes measure the QoS provided by potential parents and then select the one supporting the required QoS.

But, QoS capabilities of a node can change over time. This means that some links between nodes might not support the required QoS anymore after a certain time. Then, a new parent has to be found. On the link between the node and the parent, the initially required QoS has to be supported again. The protocol dependent and independent (layered) OM-QoS approaches were both investigated.

We evaluated all values as shown in Table 3.4 in Section 3.3.3 and additionally the NICE specific values as presented in Table 3.6.

Table 3.6: Additional Values Evaluated in Simulation Scenarios for NICE

Number of cluster mates	measures the number of cluster mates per cluster leader.
Rejoin duration	is the time a node takes to join a new cluster, if its QoS requirements are not fulfilled in its current cluster.
Node to root RTT constraints fulfilled	is the percentage of nodes for which NICE satisfies the given E2E RTT constraint during the initial join process.
Node to root RTT after join	is the RTT from a node to the multicast root directly after joining.
Node to root RTT difference	is the difference between effectively achieved node to root RTT and the node to root RTT constraint after join.

We investigated different scenarios, starting comparing the influence of the k parameter (described in Section 2.5.7) on certain issues of a native NICE network without QoS support. Therefore, we looked at the multicast fan-out, multicast hop count, node to root RTT and number of cluster mates for NICE networks with $k = 3$, $k = 4$ and $k = 5$. We then compared native NICE and QoS aware NICE (protocol dependent OM-QoS approach) using $k = 5$.

Finally, we also looked at the node to root RTT constraints explained in Section 3.2.6 that a node might request. We modified the join process to the NICE network as described in Section 3.2.6. Hence, nodes should have their node to root RTT below their requested upper bound for that value directly after they entered the network.

We performed our simulations using various network sizes with a node count from 100 to 2000 in steps of 100. Each of these steps was evaluated using 13 different distance matrices described in Section 3.3.3 in Table 3.2. For each of the

13 different distance matrices, we used the first three random seeds presented in Table 3.3 in Section 3.3.3. The seeds are influencing the arrival time of nodes, departure decisions of nodes, QoS classes of nodes, and many other random based values. Therefore, each scenario consists of 780 simulation runs. We removed 1% of the outliers (0.5% of the minimum and maximum values each) for the figures and results of the NICE evaluation presented in Section 3.4.4.

3.3.6 Chord Evaluation Scenarios

To evaluate our approach, we implemented the optimized Chord protocol (see Section 3.2.7) in the OMNet++ simulator as described in Section 3.3.3. We implemented the Chord protocol based on messages in a completely decentralized way. With some modifications, our protocol implementation in OMNet++ could also be used as an implementation of Chord for the Internet. We are assuming to have hard QoS guarantees offered by the underlying network. This can be achieved using, e.g., DiffServ or QoS provided by an approach proposed by EuQoS [68, 28, 67, 116] mechanisms. We used the protocol dependent approach of OM-QoS for Chord.

We also looked at the node to root RTT constraints explained in Section 3.2.6 that a node might request. We consider these constraints for the whole time a node is in a Chord network. Therefore, if the node to RTT changes for a node (due to changes occurring further up the multicast tree because of leaving of existing and joining of new nodes), all nodes attached to its sub-tree are also influenced by that change and might have to find a new parent. Furthermore, we limited the multicast fan-out of nodes in Chord to 7, to have a reasonable upper boundary for the fan-out.

We will look at different scenarios for evaluating Chord. First, we compare forwarder driven multicast performance using our enhanced Chord protocol with and without QoS class support. Then, we analyze the receiver driven multicast approach, where we can support also delay guarantees.

We evaluated all values as shown in Table 3.4 in Section 3.3.3 and additionally the Chord specific values as presented in Table 3.7.

Table 3.7: Additional Values Evaluated in the Simulation Scenarios for Chord

Node to root RTT Constraints Fulfilled	is the percentage of nodes for which Chord satisfies the given node to root RTT constraint.

The scenarios were evaluated using various network sizes. The different networks had a node count from 100 to 2000 in steps of 100. Each node step was evaluated using 13 different distance matrices as described in Section 3.3.3 in Table 3.2.

For each of the 13 different distance matrices, we used the first three random seeds presented in Table 3.3 in Section 3.3.3. Those random seeds influence the arrival time of nodes, departure decisions of nodes, QoS classes of nodes and other random based decisions and values. This then leads to a total of 780 simulation runs per scenario. We removed 1% of the outliers (0.5% of the min. and max. values each) for the figures and results of the Chord evaluation presented in Section 3.4.5.

3.4 Evaluation Results for OM-QoS

3.4.1 Overview

In the following Sections, we evaluate the OM-QoS enhancements to enable QoS for different P2P/ALM protocols. First, Section 3.4.2 presents the analysis and evaluation of Scribe/Pastry using Freepastry's built-in simulator. The evaluation results for CAN using the protocol independent (layered) approach is presented in Section 3.4.3. In Section 3.4.4, the evaluation results for NICE using a protocol dependent and also the protocol independent approach are presented. Finally, Section 3.4.5 presents the evaluation results for Chord, where we also evaluated the node to root RTT constraints in more detail.

3.4.2 Scribe/Pastry Evaluation Results

As we can see in Fig. 3.10 our modification of the ID assignment always performs better in respect of building QoS aware multicast trees than the random ID assignment of Pastry. As expected, the property described before holds for every end-to-end path in the multicast tree created by Scribe when using our method for assigning IDs.

Figure 3.11 shows that the maximum and average path lengths (in terms of hop counts in the Pastry P2P network) for the two methods do not significantly differ, meaning that our modification of the ID assignment method is not worse than the random ID assignment of Pastry regarding average end-to-end path lengths in multicast trees.

3.4.3 CAN Evaluation Results

Figures 3.17(a), 3.12(a), 3.13(a), 3.15(a), 3.14(a), 3.16(a), 3.18(a), 3.19(a) show the simulation results for a native CAN scenario without any QoS support but where each node gets one of 64 QoS classes assigned at startup. This allows us to still compare how well CAN by chance would build QoS enabling multicast trees. All nodes reside in one single CAN consisting of up to 2000 nodes.

In Figures 3.17(b), 3.12(b), 3.13(b), 3.15(b), 3.14(b), 3.16(b), 3.18(b), 3.19(b), we show the simulation results for an OM-QoS enabled CAN scenario with 64 QoS classes. For the OM-QoS enabled CAN scenario with 64 QoS classes, we have 64 distinct CAN layers consisting on average of 1.56 to 31.25 nodes per layer (i.e.,

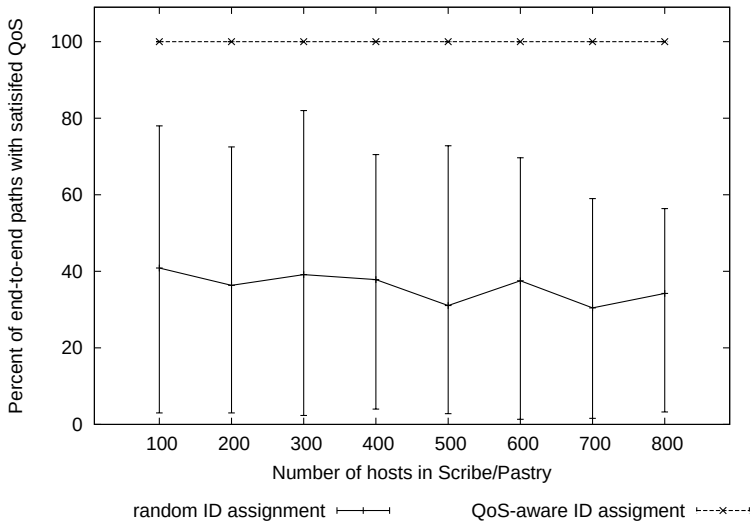


Figure 3.10: End-to-end Paths Comparison regarding QoS Satisfaction with Hard QoS and 64 QoS Classes

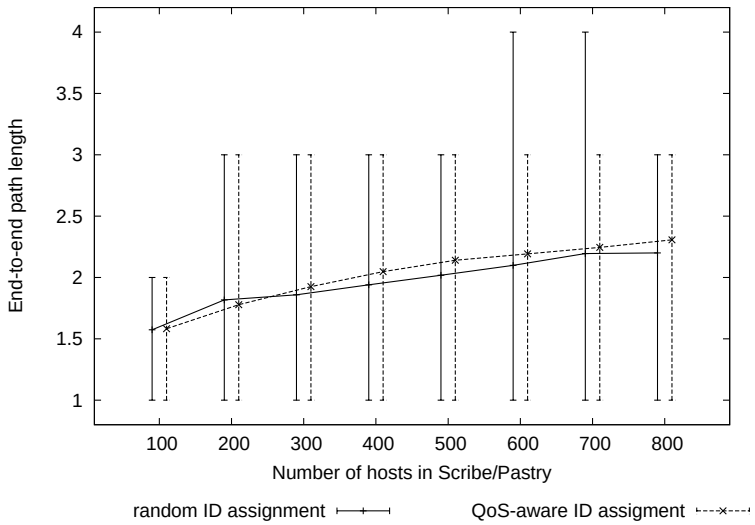
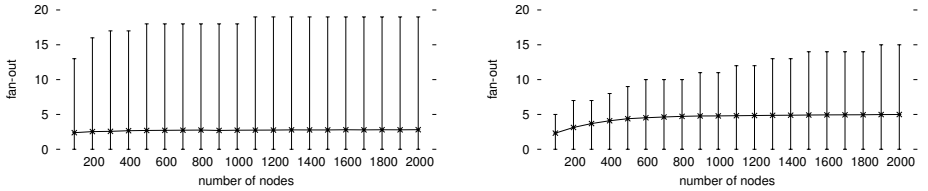


Figure 3.11: Average and Maximum Path Lengths (in Hops) for all End-to-End Paths

per QoS class) for a total number of nodes ranging from 100 to 2000 respectively. We compare the average, minimum and maximum values for the OM-QoS enabled CAN scenario with the results for the native CAN scenario without QoS support.

Multicast Fan-Out

Figure 3.12 shows the multicast fan-out, the number of direct downstream children of a node. For the native CAN scenario shown in Fig. 3.12(a), the average fan-out



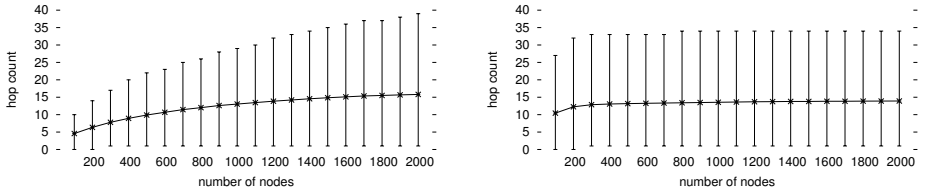
(a) Native CAN (b) OM-QoS CAN

Figure 3.12: Multicast Fan-Out in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

of 3 per node is very acceptable and remains more or less constant for the different amount of nodes. As expected, there are also leaf nodes that do not forward any messages. For the worst case, the maximum fan-out is at 19, which seems to grow only very slowly for more nodes. Looking at the OM-QoS enabled CAN scenario shown in Fig. 3.12(b), the average fan-out has been increased by 1 to 2 children. These additional children are introduced by the inter-layer forwarding mechanism. The maximum fan-out decreased because of CAN networks with smaller number of nodes on the different layers, but the average and maximum fan-out values remain almost constant when increasing the number of nodes. The minimum fan-out did not change, because there will always be pure leaf nodes.

Multicast Hop Count

In Fig. 3.13, we present the hop-count of multicast messages. The hop-count on average grows slowly when there are more nodes in a native CAN, as shown in Fig. 3.13(a). Minimum and maximum values for the hop-count behave as expected, with the minimum being more or less constant while the maximum grows steadily when more nodes are in a CAN. The average hop-count in the OM-QoS enabled



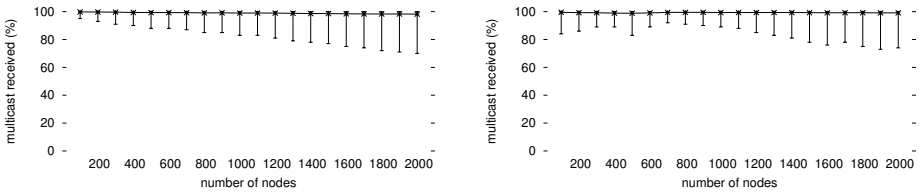
(a) Native CAN (b) OM-QoS CAN

Figure 3.13: Multicast Hop Count in Native CAN and in OM-QoS CAN with Hard QoS and 64 QoS Classes

CAN scenario as shown in Fig. 3.13(b) is generally increased due to inter-layer forwarding, but behaves now more or less constant due to smaller CAN sizes. This has a similar effect on the maximum.

Percentage of Multicast Messages Received

In Fig. 3.14, we depict the percentage of received multicast messages. More nodes in a native CAN result in less multicast messages being successfully delivered to every node, as can been seen looking at the behavior of the minimum values for the native CAN scenario presented in Fig. 3.14(a). The range for the average is be-

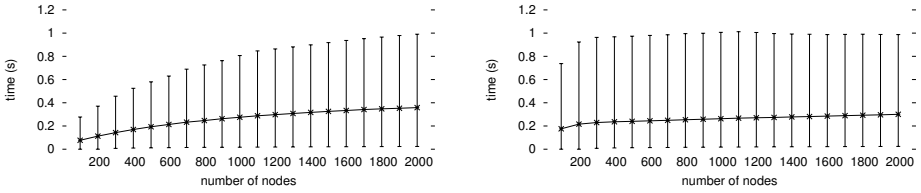


(a) Native CAN (b) OM-QoS CAN

Figure 3.14: Percentage of Multicast Messages Received in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

tween 98 to 100 percent, with a maximum of 100 percent as expected. The amount of the received multicast messages in the OM-QoS enabled CAN scenario as presented in Fig. 3.14(b) is improved, because there are only small number of nodes in a CAN and due to the reliability optimized inter-layer forwarding mechanism. The minimum value behaves less predictably, because there are now multiple CAN in parallel behaving completely different than when just having one big CAN for all nodes.

Node to Root RTT



(a) Native CAN (b) OM-QoS CAN

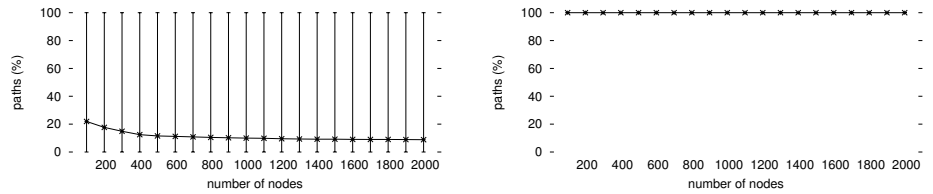
Figure 3.15: Node to Root RTT in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

Figure 3.15 shows the node to root RTT in the multicast tree. This value behaves the same as the hop-count with an average between 0.1 to 0.35 seconds and

a maximum of 1 second for the native CAN scenario as presented in Fig. 3.15(a). In the OM-QoS enabled scenario presented in Fig. 3.15(b), the RTT values receive the same impact as for hop count. They are on average a bit higher than in the native CAN scenario but behaving more or less constant with more nodes in the scenario. CAN does not offer any mechanism to optimize RTT between nodes.

Node to Root QoS

In Fig. 3.16, we present the percentage of nodes that hold the previously described property on their full path to the root of the multicast tree. Although the paths in

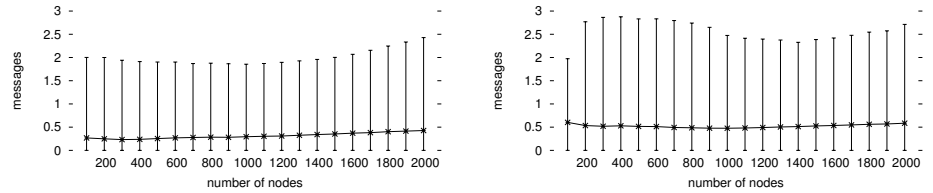


(a) Native CAN (b) OM-QoS CAN

Figure 3.16: Node to Root QoS in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

the multicast tree are not constructed in a QoS aware manner in the native CAN scenario shown in Fig. 3.16(a), on average 10 to 20 percent of the paths may hold the previously mentioned QoS property. As expected, all the paths fulfill the QoS property in the OM-QoS enabled CAN scenario shown in Fig. 3.16(b).

Average Multicast Duplicates per Multicast Message



(a) Native CAN (b) OM-QoS CAN

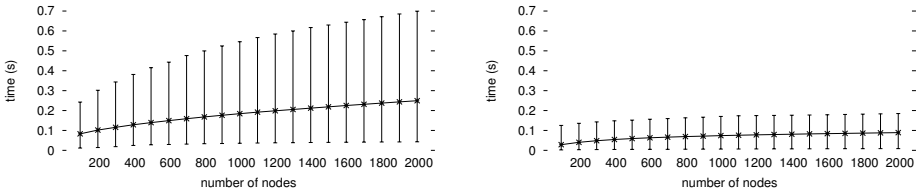
Figure 3.17: Average Multicast Duplicates per Multicast Message in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

Figure 3.17 presents the duplicates received per multicast message. The average for the native CAN scenario shown in Fig. 3.17(a) is between 0.3 to 0.5 duplicates per message, which is due to the nature of CAN. As the minimum shows, duplicates can be often avoided. For some nodes the worst case is between 2 to

2.5 duplicates per multicast message. Comparing the average with the OM-QoS enabled CAN scenario shown in Fig. 3.17(b), we see that the values behave now more or less constant due to the fact that we have smaller CAN networks. On the other hand, the average for less nodes has increased slightly, because of the duplicates that can be introduced by inter-layer forwarding. Our mechanisms between layers focuses more on reliability than on duplicate avoidance. This has the same effect on the maximum and minimum. Duplicates introduced by inter-layer forwarding could be avoided when omitting our inter-layer forwarding reliability improvements based on pro-active actions performed by inter-layer parents. Instead, nodes responsible for one QoS layer would have to explicitly subscribe to their parent nodes in other (higher) QoS layers. Those nodes would also have to explicitly react if their current parents would not forward anymore multicast messages to them. These mechanisms would omit the duplicates but could lead to one or even a chain of layers not being served anymore by multicast messages for certain periods of time.

Time Until a Node has Joined

Figure 3.18 shows the time a node required to fully join a CAN. The values in the native CAN scenario presented in Fig. 3.18(a) grow steadily with the amount of nodes, having an average ranging between 0.1 and 0.25 seconds, while the maximum is still in an acceptable range from 0.25 to 0.7 seconds. The minimum



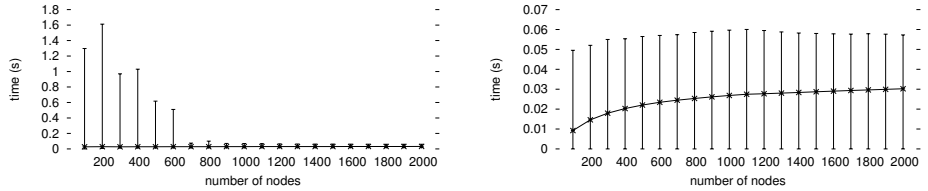
(a) Native CAN (b) OM-QoS CAN

Figure 3.18: Time Until a Node has Joined in Native CAN and in OM-QoS CAN with Hard QoS & 64 QoS Classes

remains constantly close to 0.05 seconds, which reflects the early moments of a CAN, where there are only a few nodes in the network and join related messages have only to be passed among few nodes. The OM-QoS enabled CAN scenario presented in Fig. 3.18(b) has smaller CAN in the different layer as explained in the beginning of Section 3.4.3. Therefore, joining time on average is reduced significantly. The maximum and minimum values behave accordingly.

Time Until a Node has Left

In Fig. 3.19, we present the time a node required to leave successfully. On average, nodes can quickly leave a CAN, but sometimes the handover of zones to other



(a) Native CAN (b) OM-QoS CAN

Figure 3.19: Time Until a Node has Left in Native CAN and OM-QoS CAN with Hard QoS & 64 QoS Classes

nodes may take more time if multiple nodes want to leave at the same time. When using native CAN as shown in Fig. 3.19(a), then all nodes are in only one CAN (and not distributed over multiple layers of CAN). If multiple nodes want to leave at the same time, then the coordination of zone handover can result in leaving times of up to 1.6 seconds. The leaving time in a OM-QoS enabled CAN scenario as shown in Fig. 3.19(b) is shorter. Because of the lower probability of nodes simultaneously leaving from the same CAN layer, the necessity of coordinating zone handover is less probable. Therefore, the average and maximum values grow slowly and are almost constant with more nodes being distributed evenly over the multiple layers.

Summary

Applying OM-QoS to CAN (Content Addressable Networks) enables CAN to support Quality of Service (QoS) enabled multicasting. For each QoS class, a dedicated CAN is established. These layers of CAN are then interconnected using gateway nodes. Multicast messages are then passed from a higher QoS class layer to a lower QoS layer, therefore building multicast trees that support QoS requirements. Enabling QoS for CAN using OM-QoS slightly increases the average fan-out of nodes and duplicates received per multicast message, but in consequence, this improves the percentage of multicast messages received. Other properties of CAN are not changed significantly when using OM-QoS. The overall scalability is improved, because using OM-QoS, the nodes are distributed over multiple layers of smaller CAN rather than joining one single big CAN.

3.4.4 NICE Evaluation Results

Impact of *k* Value on Native NICE

This Section compares the impact of the k value on native NICE regarding multicast fan-out, hop count, node to root RTT, and number of cluster mates. The size of a cluster varies between the lower bound k and the upper bound $3k - 1$, with k being a predefined cluster constant larger than zero, as presented in Section 2.5.7. We evaluated the maxima and minima for the different values of k individually.

But, we only show the highest maximum and lowest minimum encountered for any value of k.

In Fig. 3.20, we can see that the average fan-out is only slightly depending on k. The presented maximum is encountered using $k = 5$ and is caused by the root

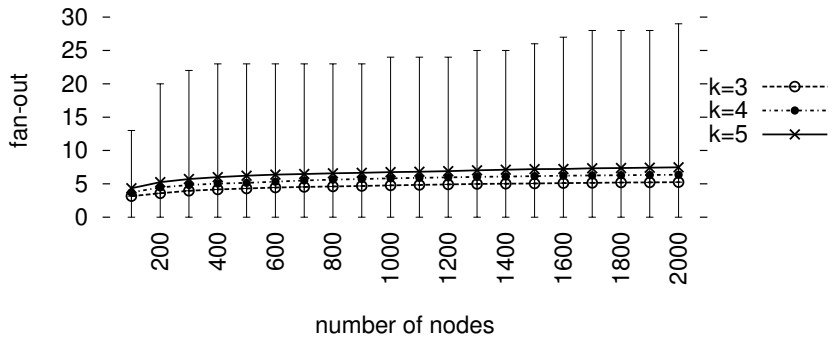


Figure 3.20: Multicast Fan-Out for Native NICE with $k = 3$, $k = 4$, and $k = 5$

node. Leaf nodes do not forward any multicast data, hence we have a minimum fan-out of 0. This is where we have the highest number of cluster mates per cluster on average, and where this particular root node has to serve every layer with multicast messages. The fan-out is directly proportional to the value of k for large NICE networks with a high number of nodes. Choosing a high value for k results in a high average fan-out.

With lower k, the hop count increases as shown in Fig. 3.21. When we have

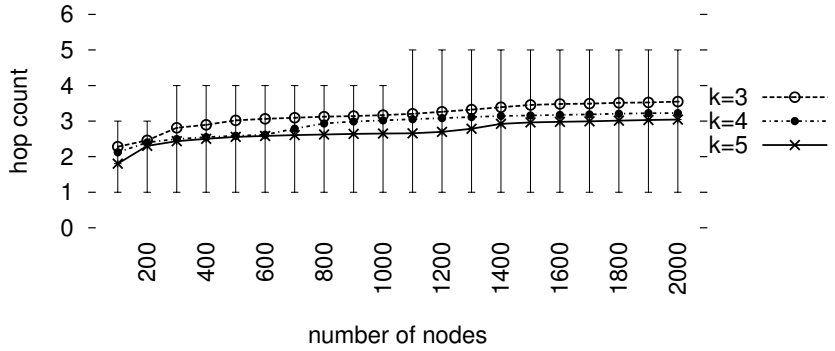


Figure 3.21: Multicast Hop Count for Native NICE with $k = 3$, $k = 4$, and $k = 5$

smaller clusters, we will get more layers, which leads to a higher hop count. The maximum is though still acceptable, only 1–2 hops higher than the average.

Since NICE always tries to optimize clusters according to the RTT between the cluster mates, the mean node to root RTT values as shown in Fig. 3.22 are only slightly increasing with lower k values. The maximum is slightly above twice the worst average.

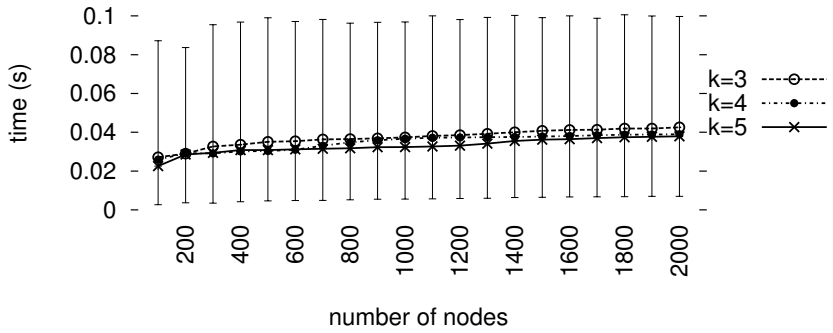


Figure 3.22: Node to Root RTT for Native NICE with $k = 3$, $k = 4$, and $k = 5$

Finally, the number of cluster mates presented in Fig. 3.23 is again heavily depending on the k value and behaving as expected, being somewhat above the effective k value. The maximum is again roughly twice the highest average value

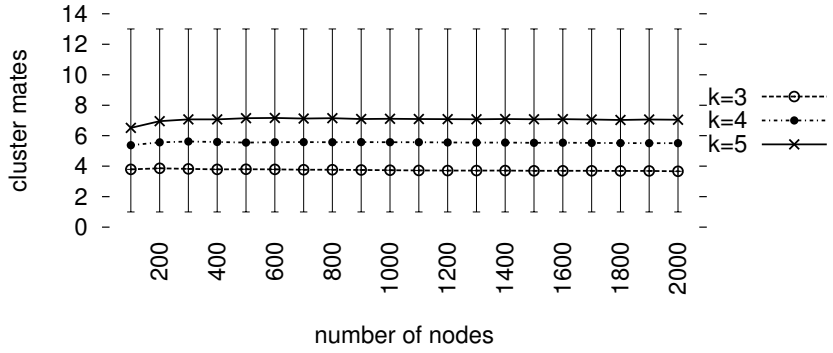


Figure 3.23: Number of Cluster Mates for Native NICE with $k = 3$, $k = 4$, and $k = 5$

encountered for $k = 5$. For all these aspects analyzed, the minima values are as expected, considering that the outliers have been removed.

NICE with Hard QoS using the Protocol Dependent Approach of OM-QoS

In this Section, we compare native NICE with QoS enabled NICE using the protocol dependent OM-QoS approach, both having $k = 5$. Choosing $k = 5$ reduces the number of layers in a NICE network, and therefore also reduces the hop count and node to root RTT while only having a slightly increased fan-out. This has been shown when determining the impact of k value on NICE.

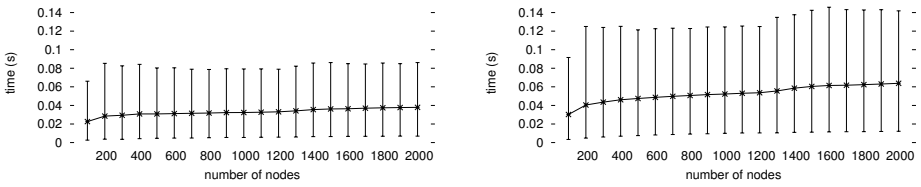
Each node gets a QoS class assigned from the range 0–255. We assume that the underlying network will provide the requested QoS for the OM-QoS enable NICE scenario. Therefore, the QoS will remain static. We call this hard QoS.

In native NICE, cluster leaders are determined using delay measurements and calculating the graph-theoretic center as described in Section 3.2.6. In OM-QoS

enabled NICE, cluster leaders are determined by the QoS class. The cluster leader of a cluster is the node having the highest QoS class in this cluster.

Since the OM-QoS enabled NICE has a different cluster leader determination mechanism not based on RTT, we compare the node to root RTT in Fig. 3.24. The native NICE mode shown in Fig. 3.24(a) is optimized and has an average node to root RTT between 20ms–40ms depending on the number of nodes in a NICE network.

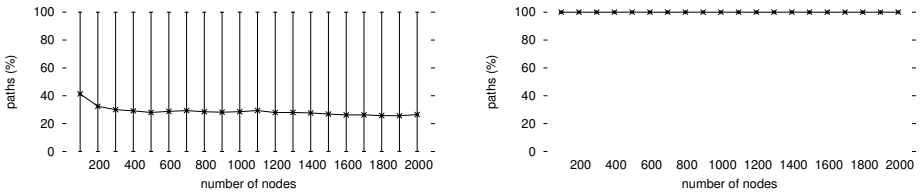
The OM-QoS enabled NICE shown in Fig. 3.24(b) has an average node to root root RTT of 30ms–60ms. Using OM-QoS adds another 50% of extra delay to the node to root RTT. This is expected due to the fact that cluster leaders are not anymore determined and optimized using delay measurements. Instead, QoS classes (i.e., bandwidth) are used to determine cluster leaders. The maximum of the node to root RTT value on the other hand almost doubles when using QoS mechanisms.



(a) Native NICE (b) OM-QoS NICE

Figure 3.24: Node to Root RTT in Native NICE and in OM-QoS NICE with Hard QoS % 256 QoS Classes

For native NICE, QoS classes are not taken into account when a node joins a NICE network or when cluster leaders have to be determined. In Fig. 3.25, we present how many paths would satisfy the property of monotonically decreasing QoS requirements as described before. In Fig. 3.25(a) we can see that on average,



(a) Native NICE (b) OM-QoS NICE

Figure 3.25: Node to Root QoS in Native NICE and in OM-QoS NICE with Hard QoS & 256 QoS Classes

only 30% of the paths hold the property for native NICE, whereas in Fig. 3.25(b) with OM-QoS enabled NICE, 100% of the paths fulfill QoS.

Still, introducing QoS to NICE using OM-QoS allows us to guarantee that all

paths from the root node to the leaf nodes in the multicast tree fulfill the QoS requirements while adding an acceptable overhead in terms of delay.

NICE with Soft QoS using the Protocol Dependent Approach of OM-QoS

In this Section, we look at the scenario with soft QoS. This means that the QoS guarantees can change over time. The difference between hard QoS and soft QoS is described in Section 3.2.1. If QoS cannot further be supported along a path, the node has to look for a new parent (or cluster leader in NICE) that actually supports again its QoS requirements. In this evaluation, there is a 50% chance for every node that its QoS requirements are not supported anymore during the time interval of 10s to 100s after joining the NICE network. Again, we use 256 QoS classes and also $k = 5$ for this scenario with the protocol dependent OM-QoS approach to support QoS for NICE.

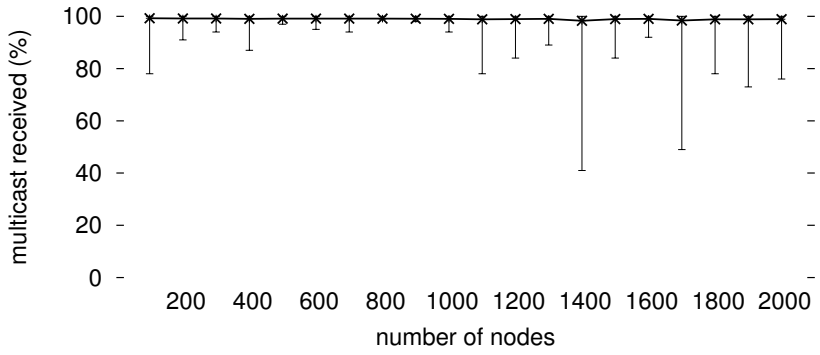


Figure 3.26: Percentage of Multicast Messages Received in OM-QoS NICE with Soft QoS & 256 QoS Classes

As shown in Fig. 3.26, the dynamic behavior of the QoS guarantees does not have a negative impact on the percentage of multicast messages received by nodes on average. Almost all multicast messages are delivered as intended. Some nodes though might have a higher loss rate due to multiple rejoins, merging and split operations, cluster leader and root transfers, as visible from the minimum value. During these times, there might be some moments where a node is not part of a cluster, and therefore might not receive multicast messages from a cluster leader. This is especially the case for nodes that are only participating in a NICE network for a short time. If during this time the previously mentioned operations (rejoin, split, merge, transfers) occur frequently, it can happen that such nodes only receive a few multicast messages. This then leads to a high loss rate.

The number of cluster mates presented in Fig. 3.27 is lower (by one node) than in the native NICE network previously shown in Fig. 3.23 for $k = 5$. This is due to the fact that nodes actually leave their cluster and rejoin other clusters which even has a positive impact. Such a behavior contributes to a more equal distribution of nodes among the layers and clusters.

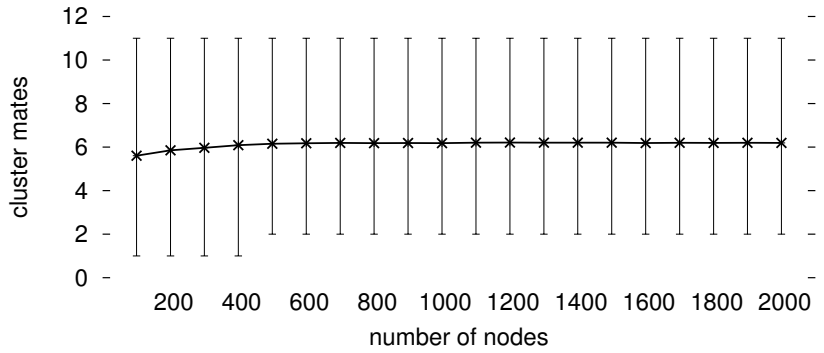


Figure 3.27: Number of Cluster Mates in OM-QoS NICE with Soft QoS & 256 QoS Classes

The rejoin duration is presented in Fig. 3.28, where the average rejoin duration is around 20ms and the maximum is around 50ms, which are both acceptable values.

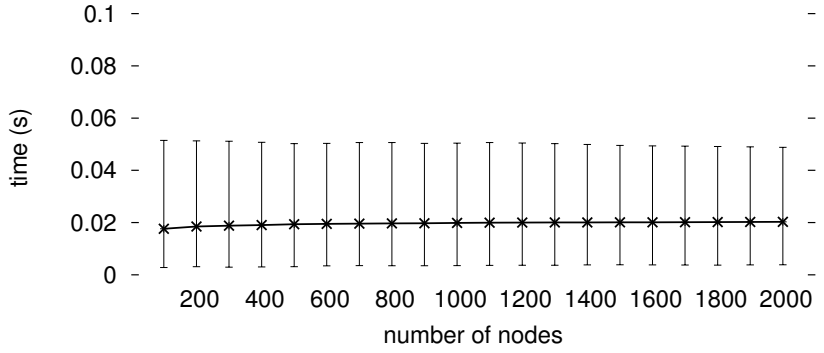


Figure 3.28: Rejoin Duration in OM-QoS NICE with Soft QoS & 256 QoS Classes

Figure 3.29 presents how many paths actually fulfill certain node to root RTT constraints for an OM-QoS enabled NICE network that however does not take these RTT constraints into account. Each node selects a node to root RTT constraint in the range of 25ms–50ms. We then check how many paths actually fulfill these constraints during the time a node participates in the NICE network.

NICE with Hard QoS using the Protocol Independent Approach of OM-QoS

The results for a QoS aware NICE network using the protocol independent (layered) OM-QoS approach with 32 QoS classes and $k = 5$ are presented in this Section. Native NICE is heavily optimizing RTT among cluster mates, and therefore also indirectly optimizes node to root RTT. Applying the protocol independent OM-QoS approach to NICE using 64 or even 256 QoS classes would have a strong impact on the hop count and ultimately also on the node to root RTT. Therefore, to

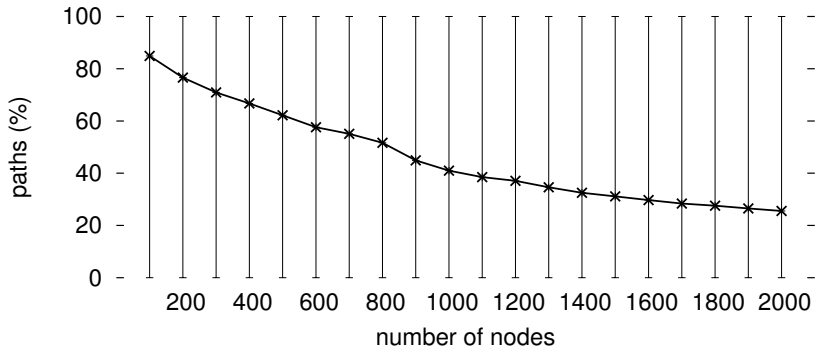


Figure 3.29: Node to Root RTT Constraints Fulfilled in OM-QoS NICE with Soft QoS & 256 QoS Classes

keep the overall hop count and node to root RTT still on an acceptable level compared to native NICE, we decided to only use 32 QoS classes instead of 64 or 256 QoS classes as used for the evaluations of CAN in Section 3.4.3 and also earlier in this section. Nodes are now distributed over 32 different NICE networks (slices). There is one dedicated NICE network for each of the 32 QoS classes.

Before, there was only one NICE network for all the nodes independent of their QoS class. Nodes joining the same network results in more layers and clusters in the network compared to distributing them among several slices, which then have less layers and clusters.

We now used 32 QoS classes, which is still a reasonable value, to have an acceptable hop count compared to native NICE, because of the additional hops introduced by forwarding data between slices. As already mentioned, we now have much smaller NICE networks in terms of participants, with less clusters and less layers. Nodes that join a specific NICE network slice might be distributed such that only small clusters would be formed in those NICE network slices.

Also less layers in such a NICE network slices might be necessary due to a much lower number of participating nodes per NICE network slice compared to one big NICE network where all nodes would join. This fact leads to a lower fan-out per node, less hops to the root of the multicast tree and also a lower node to root RTT inside the individual NICE network slices compared to having one big NICE network holding all nodes.

The average fan-out presented in Fig. 3.30 is between 3–8 hops. Although we have additional fan-out caused by the gateway links, the average fan-out starts lower than for the native NICE with $k = 5$ as shown in Fig 3.20. When the number of nodes increases, the average though starts to remain constant and is similar to the average fan-out in native NICE. The maximum is though raising almost up to 20, which is due to the multiple optimized gateway links as well as the backup links (when they are in use).

Using multiple slices of NICE in parallel has an impact on the average hop count as presented in Fig. 3.31. We are now using 32 slices of NICE in parallel.

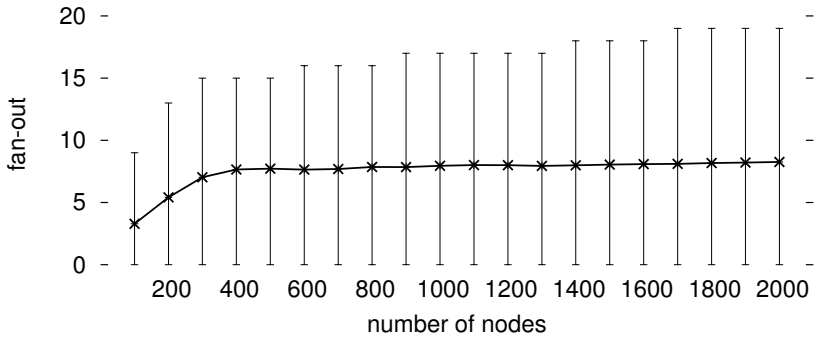


Figure 3.30: Multicast Fan-Out in OM-QoS NICE using the Protocol Independent Approach with Hard QoS & 32 QoS Classes

These NICE network slices have to be interconnected as described in Section 3.2.6 which causes additional hops. On average, there are now between 8–9 hops, with the maximum raising almost up to 20 hops. Compared to the hop count in native NICE presented in Fig. 3.21 for $k = 5$, the average value for the hop count is three times higher (around 9 hops instead of 3 hops) and the maximum hop count is almost 4 times higher.

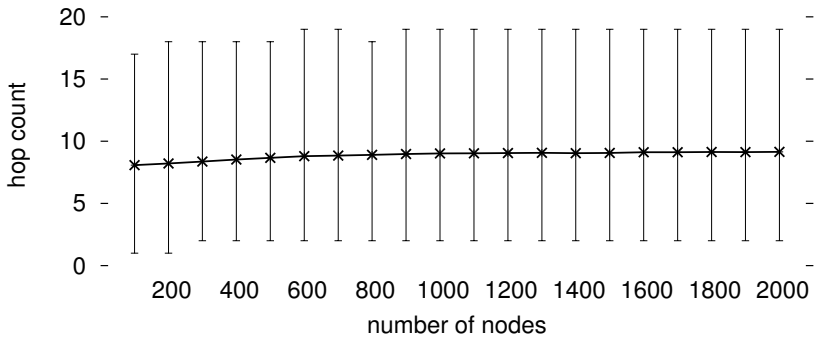


Figure 3.31: Multicast Hop Count in OM-QoS NICE using the Protocol Independent Approach with Hard QoS & 32 QoS Classes

As shown in Fig. 3.32, the average node to root RTT suffers from the same penalty of having multiple instances. Having additional hops results in a higher node to root RTT, which is now ranging from 110ms–120ms. Also, the maximum node to root RTT increases up to 400ms. Compared to the protocol dependent approach of OM-QoS applied to NICE as presented in Fig. 3.24(b), the average as well as the maximum node to root RTT are around 2–3 times higher. Comparing it to the same native NICE results presented in Fig. 3.24(a), the average and maximum node to root RTT values are even between 3–5 times higher.

Having multiple parallel instances has again an impact on the number of average cluster mates as presented in Fig. 3.33. Again, the nodes are distributed over

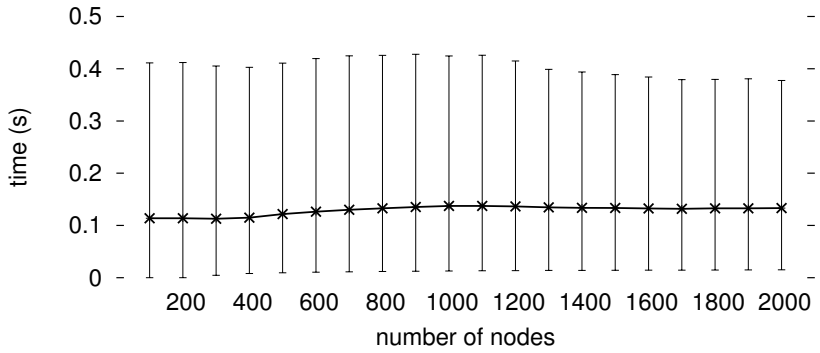


Figure 3.32: Node to Root RTT in OM-QoS NICE using the Protocol Independent Approach with Hard QoS & 32 QoS Classes

parallel instances of NICE. Therefore, the average number of cluster mates is reduced compared to the native NICE network with $k = 5$ as presented in Fig. 3.23. The average number of cluster mates for the protocol dependent approach of OM-QoS with NICE as presented in Fig. 3.27 is almost constant at 6. Therefore, for small numbers of nodes in a NICE network, the protocol independent approach creates generally smaller cluster sizes. For scenarios with more than 1400 nodes, the average number of cluster mates starts to be similar for both OM-QoS approaches. Although we have now 32 NICE instances in parallel, the overall resulting values

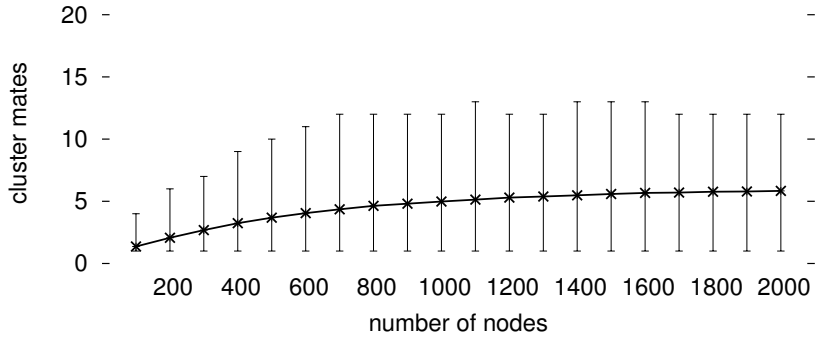


Figure 3.33: Number of Cluster Mates in OM-QoS NICE using the Protocol Independent Approach with Hard QoS & 32 QoS Classes

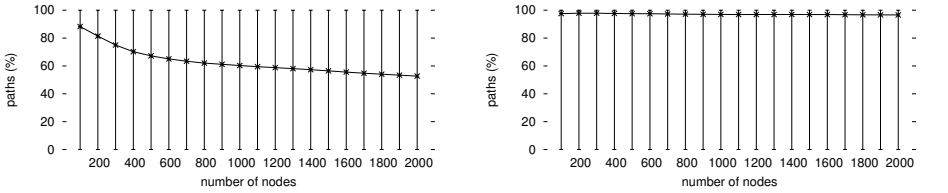
from that scenario remain acceptable. In some cases they even have been improved, e.g., average number of cluster mates and maximum fan-out.

NICE with Node to Root RTT Constraints and Hard QoS using the Protocol Dependent Approach of OM-QoS

We now present the evaluation results comparing QoS aware NICE using the normal join process of NICE and a modified (RTT constraints aware) join process

presented in Section 3.2.6 that takes the node to root RTT constraints into account. When nodes join the NICE network, they select randomly a node to root RTT constraint in the range of 25ms–50ms. The average hop delay using the distance matrices from Table 3.2 in Section 3.3.3 is around 25ms. This limits the average hop count between 1 to 2 hops when nodes should have their node to root RTT constraints fulfilled.

Figure 3.34 presents the number of fulfilled node to root RTT constraints for a QoS aware NICE network. The values using a join process that does not take

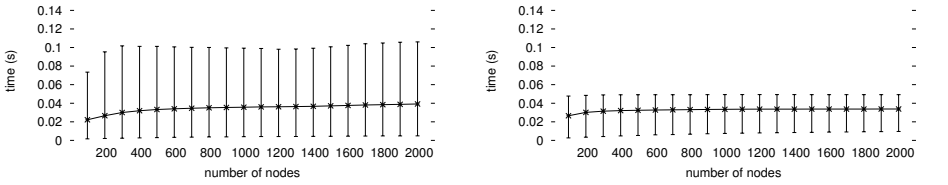


(a) OM-QoS NICE, Normal Join (b) OM-QoS NICE, RTT Constraints Aware Join

Figure 3.34: Node to Root RTT Constraints Fulfilled for Normal Join and for RTT Constraints Aware Join in OM-QoS using Hard QoS with the Protocol Dependent Approach

those node to root RTT constraints into account is presented in Fig. 3.34(a).Almost half of the nodes do not have a node to root RTT, which is below their required constraints (directly after joining the NICE network). In a NICE network with the modified join process taking these constraints into account, almost all nodes have their RTT to root constraints satisfied (directly after joining the NICE network) as shown in Fig. 3.34(b).

Figure 3.35 shows the node to root RTT after a node has joined a NICE network. The resulting node to root RTT after a node has joined a NICE network using



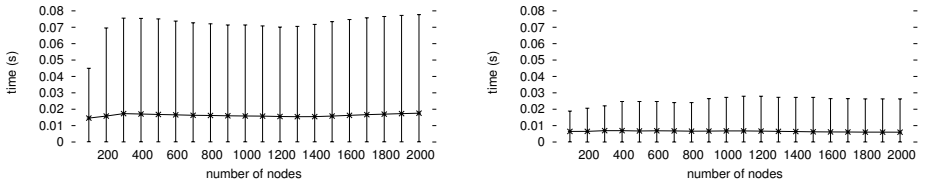
(a) OM-QoS NICE, Normal Join (b) OM-QoS NICE, RTT Constraints Aware Join

Figure 3.35: Node to Root RTT after Join for Normal Join and for RTT Constraints Aware Join in OM-QoS using Hard QoS with the Protocol Dependent Approach

the normal NICE join process is presented in Fig. 3.35(a). The average is close to the upper end (50ms) of the node to root RTT constraint's range with more nodes in the network. The maximum of the node to root RTT is around 100ms, which is the double of the range. This means that the nodes, which have their node to root RTT constraints not fulfilled have a node to root RTT, which is often significantly

above what they actually requested. For applications with tight real-time requirements, such as MMOGs, these high differences would not be tolerable. The results for a NICE network using the modified join process, which tries to fulfill the node to root RTT constraints (between 25ms–50ms) of a node are shown in Fig. 3.35(b). The average for the node to root RTT is close to the middle (37.5ms) of the RTT constraint's range. The maximum is at the end of the constraint's range (50ms). This shows that the nodes not having their node to root RTT constraints fulfilled are normally just slightly above the node to root RTT value they requested. For most applications with tight realtime requirements, such minor differences are still tolerable.

In Figure 3.36(b), the difference from the required node to root RTT constraint directly after joining is presented. As expected, using the normal NICE join process



(a) OM-QoS NICE, Normal Join (b) OM-QoS NICE, RTT Constraints Aware Join

Figure 3.36: Node to Root RTT Difference for Normal Join and for RTT Constraints Aware Join in OM-QoS using Hard QoS with the Protocol Dependent Approach

presented in Fig. 3.36(a) results in a higher difference to the node's RTT constraints compared to the modified join process presented in Fig. 3.36. The node to root RTT values achieved using the modified join process would be normally still tolerable. This is due to the fact that the differences are small and the achieved values for the node to root RTT are generally only slightly above the constraint. Keep in mind that using the modified join process results in only a few nodes that do not have fulfilled their requirements regarding the node to root RTT constraint (directly after joining the NICE network) as shown in Fig. 3.34(b).

Summary

We applied the protocol dependent and protocol independent approaches of OM-QoS to NICE and evaluated it using various hard QoS and soft QoS scenarios.

Using the protocol dependent approach introduces a slight additional delay compared to native QoS-unaware NICE. This is due to the modified cluster leader determination mechanism, but does not change other basic aspects of NICE significantly. All paths from the nodes to the root of the multicast tree in the hard QoS and soft QoS scenarios support QoS when OM-QoS was used.

The soft QoS results show that in dynamic QoS environments, nodes quickly can find new clusters (which means parents in the multicast tree) in case QoS is not

supported anymore by its current cluster. In the hard QoS scenarios, QoS is statically guaranteed by resource reservation performed in the underlaying network. Hence, joining a cluster that supports the QoS requirements of a node means that the cluster leader / parent will always support the requested QoS as long as the node remains in the NICE network.

Using the protocol independent (layered) approach with NICE introduces additional hops and has higher delays compared to the protocol dependent approach. But, using the protocol independent approach also reduces other aspects, such as average number of cluster mates and maximum fan-out. This is due to the fact that there are now smaller NICE networks existing and cooperating in parallel slices.

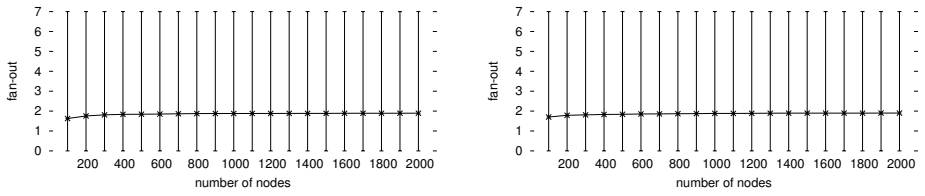
Finally, we presented an approach that takes also node to root RTT constraints of joining nodes into account. When a node joins a NICE network, it tries to find a cluster that supports its node to root RTT requirements. A joining mechanism that takes node to root RTT constraints into account enables that the majority of the nodes have those constraints fulfilled directly after joining.

3.4.5 Chord Evaluation Results

Forwarder Driven Multicast using Enhanced Native Chord and OM-QoS Chord with Hard QoS Support

We first compare the forwarder driven multicast approach (see Section 2.3.6) of our enhanced Chord protocol (as described in Section 3.2.7) without QoS (enhanced native Chord) and with QoS (OM-QoS Chord).

In Fig. 3.37, we present the multicast fan-out. Comparing Figures 3.37(a) and 3.37(b), we can see that the fan-out behaves equally whether QoS is enabled using OM-QoS or not enabled just using enhanced native Chord. The average fan-out is

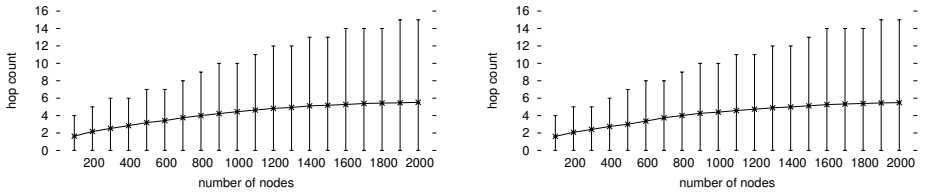


(a) Enhanced Native Chord (b) OM-QoS Chord

Figure 3.37: Multicast Fan-Out using Forwarder Driven Multicast for Enhanced Native Chord and for OM-QoS Chord with Hard QoS & 256 QoS Classes

constantly around 2. The maximum fan-out is always at 7, because our enhancements of Chord presented in Section 3.2.7 limit the maximum fan-out to 7. Hence, our enhanced Chord (as described in Section 3.2.7) is very scalable in terms of large networks. Leaf nodes do not forward and multicast data. Therefore, the minimum fan-out value is at 0. As a conclusion, enabling QoS support for Chord does not change the properties regarding fan-out

In Fig. 3.38, we present the multicast hop count. There is not a significant difference between the multicast hop count for enhanced native Chord as in Fig. 3.38(a) and for OM-QoS Chord as shown in Fig. 3.38(b). The average of hops is between

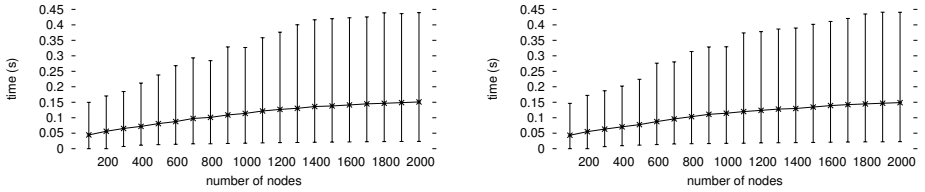


(a) Enhanced Native Chord (b) OM-QoS Chord

Figure 3.38: Multicast Hop Count using Forwarder Driven Multicast for Enhanced Native Chord and for OM-QoS Chord with Hard QoS & 256 QoS Classes

2 to 6 and the maximum raises from 4 to 15. The node to root RTT correlates with the multicast hop count.

Figure 3.39 shows the node to root RTT. Comparing Figures 3.39(a) and 3.39(b) shows that the average node to root RTT is between 50 to 150 ms for both enhanced native Chord as well as OM-QoS Chord. The maximum of the node to root RTT

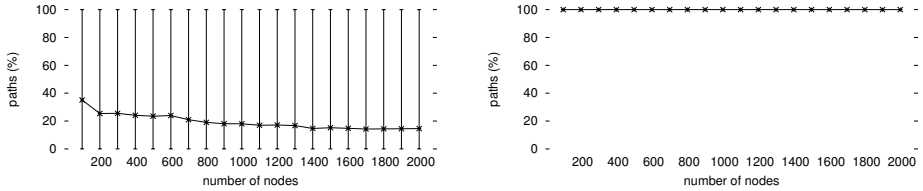


(a) Enhanced Native Chord (b) OM-QoS Chord

Figure 3.39: Node to Root RTT using Forwarder Driven Multicast for Enhanced Native Chord and for OM-QoS Chord with Hard QoS & 256 QoS Classes

starts for both versions of Chord at 150 ms and increases up to 450 ms. As a conclusion, modifying the ID assignment method to enable QoS does not change the properties of Chord regarding multicast hop count and node to root RTT.

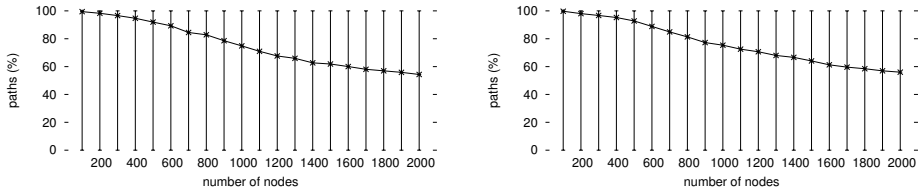
In Fig. 3.40, we present the percentage of node to root QoS fulfilled. Nodes assign themselves a QoS class from the range 0–255. We check how many paths from the root to all nodes hold the previously described QoS path properties. Figure 3.40(a) presents the results for enhanced native Chord using random ID assignment, i.e., QoS classes are not taken into account when node IDs are assigned. Here, only 15% to 40% of the paths hold the QoS path properties. Since at some times only a few nodes could remain in a Chord network, there can be moments where all paths or no paths at all fulfill the QoS path properties. On the other hand, with OM-QoS Chord, 100% of the paths fulfill the QoS path properties. This is shown in Fig. 3.40(b).



(a) Enhanced Native Chord (b) OM-QoS Chord

Figure 3.40: Node to Root QoS using Forwarder Driven Multicast for Enhanced Native Chord and for OM-QoS Chord with Hard QoS & 256 QoS Classes

In Fig. 3.41, we present the percentage of node to root RTT constraints fulfilled. Nodes also assign themselves a node to root RTT constraint from the range of 100–200ms. The forwarder driven multicast approach does not offer a mechanism to fulfill those constraints. Therefore, as shown in Figures 3.41(a) and 3.41(b), paths



(a) Enhanced Native Chord (b) OM-QoS Chord

Figure 3.41: Node to Root RTT Constraints Fulfilled using Forwarder Driven Multicast for Enhanced Native Chord and for OM-QoS Chord with Hard QoS & 256 QoS Classes

from the root to nodes do not always satisfy the constraints. In small networks, the constraints are easily met, but with larger networks and increasing hop count, the average value goes down below 60%. There is no significant difference between OM-QoS Chord and enhanced native Chord. The minimum of 0% and the maximum of 100% for the node to root RTT constraints fulfilled are due to the fact that sometimes only a few nodes could remain in a Chord network. Hence, in those moments either all paths or no paths at all fulfill the node to root RTT constraints, resulting in the mentioned values for the minimum and maximum node to root RTT constraints fulfilled.

Receiver Driven Multicast for Chord with Hard QoS Support

Using the receiver driven multicast approach presented in Section 2.3.6, we do not only take the QoS class mechanism into account when joining Chord, but also the node to root RTT constraints when looking for a multicast parent. These node to root RTT constraints of nodes range from 100–200ms. We determined this range by analyzing the average overall hop count ($\sim$ 4 hops) presented in Fig. 3.38(a) in relation to the average RTT ($\sim$ 25ms) between nodes as shown in Table 3.2.

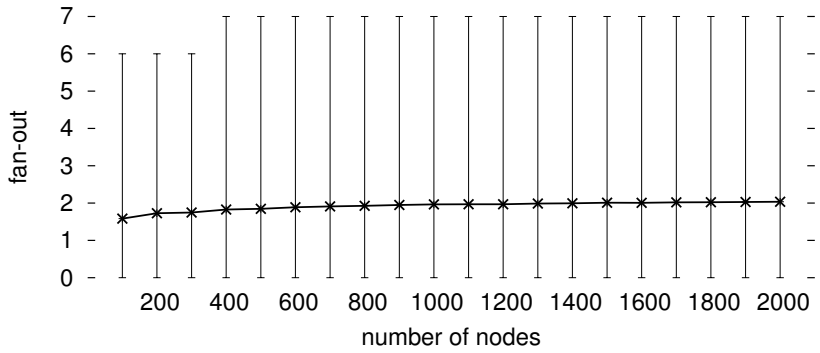


Figure 3.42: Average Multicast Fan-Out using Receiver Driven Multicast for Chord with Hard QoS Support

First, we will look at the multicast fan-out. Figure 3.42 shows that the multicast fan-out for receiver driven multicast behaves similarly as the multicast fan-out of forwarder driven multicast presented in Figures 3.37(a) and 3.37(b). The average multicast fan-out is almost identical for forwarder and receiver driven multicast. The maximum multicast fan-out is also constant at 7 for networks with more than 400 nodes.

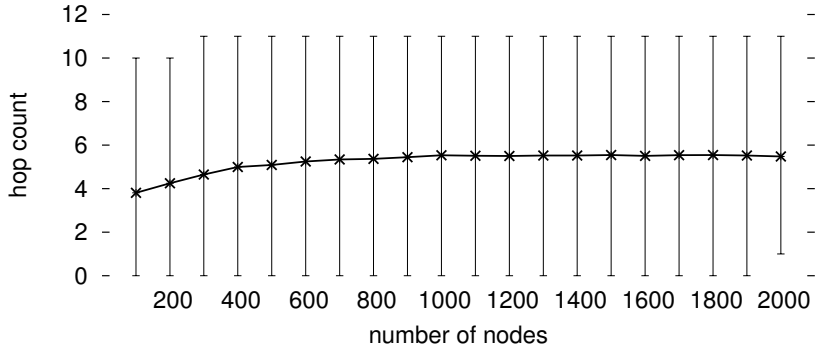


Figure 3.43: Average Multicast Hop Count using Receiver Driven Multicast for Chord with Hard QoS Support

Next, we look at the multicast hop count for receiver driven multicast in Chord. The average multicast hop count presented in Fig. 3.43 is between 4 and 5. This is slightly higher than for forwarder driven multicast as presented in Figures 3.38(a) and 3.38(b). In the forwarder driven multicast approach, the root or a node with a low ID can already reach some nodes at the upper end of the ID space via one hop. This is due to the multicast forwarding mechanism using the finger table to determine children nodes for a parent. But, this is not the case for receiver driven multicast, where the child determines and selects its parent. Here, a child node tries to find a potential parent in its ID neighborhood, from the range $[\frac{nodeID}{2}, nodeID]$. This limits the distance between hops in terms of ID space and enforces nodes to

find their parent nodes in their ID neighborhood. On the other hand, this has a positive impact on the maximum multicast hop count value. In Figures 3.38(a) and 3.38(b) compared to Fig. 3.43, the hop count starts at a larger value but grows more slowly and only up to 11 hops. This means that the receiver driven multicast approach scales better in terms of multicast hop count than the forwarder driven multicast approach. The minimum multicast hop count value of 1 with 2000 nodes is solely due to the outlier removal.

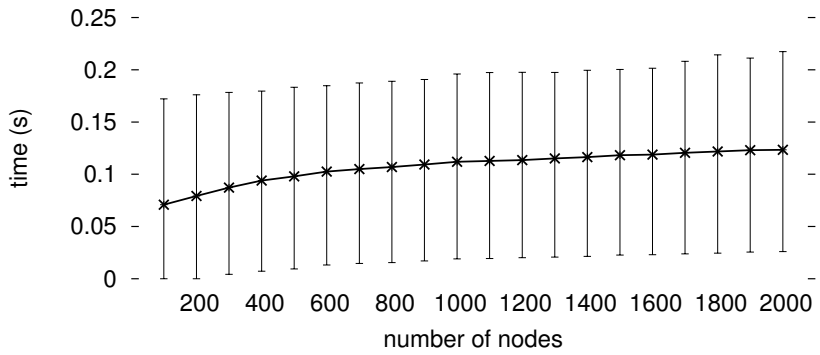


Figure 3.44: Average Node to Root RTT using Receiver Driven Multicast for Chord with Hard QoS Support

The node to root RTT results for receiver driven multicast are shown in Fig. 3.44. The forwarder driven multicast approach results were shown in Figures 3.39(a) and 3.39(b). Comparing them with the receiver driven multicast approach shows that the average node to root RTT starts at a higher value but grows more slowly. It is only in the range of 70–125ms compared to 50–150ms for the forwarder driven multicast approach. Also the maximum node to root RTT is lower and grows very slowly. This behavior is of course due to the fact that we take the node to root RTT constraint of nodes into account when they look for a multicast parent. The maximum node to root RTT value starts below the upper boundary of the constraints range (100–200ms) with a low number of nodes in the network.. With more nodes in the network, the maximum though exceeds the upper boundary of 200ms. But still, the receiver driven multicast approach is more scalable in terms of node to root RTT value than the forwarder driven multicast approach.

The percentage of node to root RTT fulfilled for receiver driven multicast is presented in Fig. 3.45. The receiver driven multicast approach performs much better than the forwarder driven multicast approach presented in Figures 3.41(a) and 3.41(b). For network sizes up to 1200 nodes, the average percentage is above 95% for the receiver driven multicast approach. Afterwards, it falls down just slightly below 92% for up to 2000 nodes. This is significantly better than using the forwarder driven multicast approach, where the average percentage is below 68% for networks with 1200 nodes. It even falls below 55% for networks of 2000 nodes.

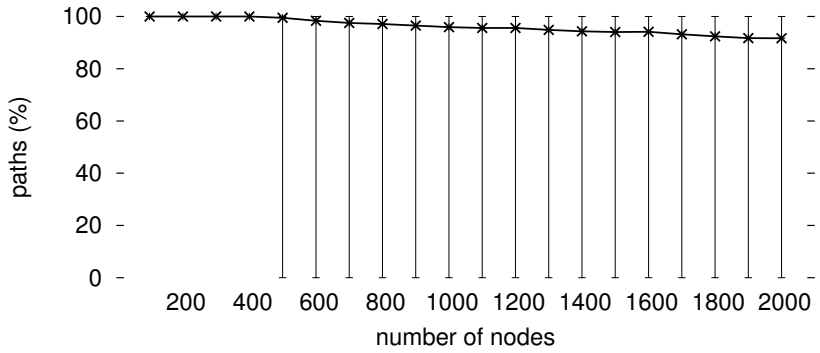


Figure 3.45: Average Node to Root RTT Constraints Fulfilled using Receiver Driven Multicast for Chord with QoS Support

Summary

All our optimizations considered, the forwarder driven multicast approach in Chord can provide a very well distributed multicast tree. The modified Chord is very reliable (100% received multicast messages). It is also very scalable, since we have put a multicast fan-out limit of 7. Generally, it behaves very robust, even in moments with constant rejoins. The overall latencies are quite good due to a reduced hop count caused by a well balanced tree. Multicast trees built using the receiver driven multicast approach can provide multicast node to root RTT guarantees. Almost all paths are fulfilling the multicast node to root RTT constraints of nodes for small and medium sized networks. For large networks of up to 2000 nodes, a very high percentage (down to 92%) of the paths still fulfill the constraints. This receiver driven multicast approach also reacts to RTT changes of nodes over time and adapts and rearranges the multicast tree to again fulfill the constraints. Generally, the receiver driven multicast approach seems to scale even better than the forwarder driven multicast approach.

3.4.6 Summary of Evaluation Results

In Table 3.8, we present a summary of the evaluation results for Scribe/Pastry, CAN, NICE and Chord. We present the results of the native protocol versions as well as of the OM-QoS enabled protocol versions using either Hard-QoS or Soft-QoS. We applied and evaluated the protocol independent approach (PIA) of OM-QoS for CAN as well as for NICE, and used the protocol dependent approach (PDA) of OM-QoS to evaluate Scribe/Pastry, NICE and Chord. We also evaluated OM-QoS NICE with Hard-QoS and node to root RTT constraints support (E2ERTT) when nodes join a NICE network. For Chord, we used forwarder driven multicast (FDM) to evaluate native Chord and OM-QoS Chord without node to root RTT constraints support. We evaluated OM-QoS Chord with full-time node to root RTT constraints support using receiver driven multicast (RDM).

Table 3.8: Summary of Evaluation Results for Scribe/Pastry, CAN, NICE and Chord

Native Scribe / Pastry	Native Scribe/Pastry can only by chance support QoS requirements of nodes for 30–40% of end-to-end paths.
PDA OM-QoS Scribe / Pastry	Applying OM-QoS to Scribe/Pastry in order to have 100% of end-to-end paths support QoS does not significantly change the average hop count.
Native CAN	The average and especially maximum fan-out and hop count of nodes is very high when comparing it to Scribe/Pastry, NICE and Chord. Native CAN does only enable 10–20% of the end-to-end paths to support QoS.
PIA OM-QoS CAN, Hard-Qos	Applying OM-QoS with the protocol independent approach to CAN adds some overhead (0–100%) to the average fan-out and hop-count, but has lower maxima for those values. With increasing number of nodes, the overhead decreases and becomes almost negligible. The percentage of multicast messages received does not differ significantly to native CAN, but all paths support QoS.
Native NICE	The average node to root RTT in native NICE is much lower than in CAN or Chord, whereas the fan-out is slightly higher than in Chord, but much better than in CAN.
PDA OM-QoS NICE, Hard-Qos	Applying OM-QoS to NICE guarantees that 100% of the end-to-end paths support QoS whereas native NICE only supports between 20–40% of the paths to support QoS. Using OM-QoS introduces roughly 50% overhead to the node to root RTT, but does not change the fan-out.
PDA OM-QoS NICE, Soft-Qos	Due to the fact that nodes sometimes have to rejoin in order to have their QoS requirements satisfied, the average cluster size is slightly lowered when using OM-QoS in Soft-QoS environments. The rejoin time of such nodes is very acceptable (around 20ms).
PIA OM-QoS NICE, Hard-Qos	The protocol independent approach of OM-QoS for NICE increases significantly fan-out, hop-count and node to root RTT, but scales better with higher number of nodes in the network.
PDA OM-QoS NICE, Hard-Qos, E2ERTT	Introducing node to root RTT constraints support when nodes join a NICE network results in almost 100% of the paths fulfilling these constraints. Native NICE can only support node to root RTT constraints for 60–90% of the paths. The differences between achieved and required node to root RTT for paths not fulfilling the node to root RTT constraints are higher in native NICE.

Native Chord, FDM	Native Chord has a very low average fan-out due to our optimizations, but only 20–40% of paths support QoS.
PDA OM-QoS Chord, Hard-Qos, FDM	Applying OM-QoS to Chord does not significantly change fan-out, hop count or node to root RTT values but enables all paths to support QoS.
PDA OM-QoS Chord, Hard-Qos, E2ERTT, RDM	Using receiver driven multicast with OM-QoS Chord adds some overhead to the hop count for lower number of nodes in the network, but enables 90–100% of the paths to fulfill node to root RTT constraints (compared to 60–100% for forwarder driven multicast without node to root RTT constraints support).

In general, we can say that using OM-QoS normally enables all paths to support QoS. Applying OM-QoS to a P2P/ALM protocol can introduce some overhead, which is though often negligible depending on the application scenario. The gain of having QoS support surely outweighs the potential overhead.

Using OM-QoS with Hard-QoS or Soft-QoS does not really make such a difference. The difference between protocol dependent and protocol independent approach is often noticeable for networks with low number of nodes, but starts to be negligible with higher number of nodes in the network. The protocol independent approach generally scales better than the protocol dependent approaches.

3.5 Conclusion

In this Chapter, we presented OM-QoS (Quality of Service for Overlay Multicast). OM-QoS is self-managing middleware, which supports efficient and reliable usage of content distribution networks based on Application Layer Multicast (ALM). It manages the Quality of Service (QoS) requirements of nodes and enables all multicast paths in an ALM, from the root to leaf nodes, to hold certain QoS constraining properties.

We presented a protocol dependent approach, which modifies the P2P/ALM protocols in order to support QoS. For Scribe/Pastry and Chord, mapping of nodes to the ring is based on the nodes' QoS. Depending on the distribution of nodes to QoS classes, this could also lead to a potential problem with uneven distribution of nodes on the ring. This could be solved by introducing adaptive and dynamic QoS to ID mapping. A Scribe/Pastry or Chord ring could be periodically restructured depending on the range of QoS classes used. If some QoS classes in a sub-range would not be actively used by any nodes, the corresponding segments in the ring could also be omitted and be used by nodes actually having other (higher or lower) QoS classes. This would help to use the whole ID range more evenly and would also better balance the distribution of nodes on the Scribe/Pastry or Chord ring.

The protocol dependent OM-QoS approach assumes an underlying multicast tree, but is not limited to tree-based ALMs. Also mesh-based ALMs could be supported. Ultimately, using a receiver driven approach as presented for Chord would also enable QoS in such mesh-based systems. To still support the robustness of such mesh-based systems, backup or parallel links to multiple parents could be established. A link prioritization mechanism would though be required to avoid duplicates.

Furthermore, we also presented a protocol independent approach, which can be used with potentially any P2P/ALM protocol. The protocol independent approach would also work with mesh-based overlay networks, because the individual P2P/ALM instances for each QoS class layer do not need to support/offer a tree-like structure

Our evaluations show that using OM-QoS, these protocols support QoS on all paths from the root node to leaf nodes by only introducing a slight overhead compared to the QoS-unaware protocol versions in terms of multicast hop count, fan-out, node to root RTT, etc. We learned that is not too difficult to support our introduced QoS class mechanism for different P2P/ALM protocols, but that supporting node to root RTT constraints cannot be achieved by all P2P/ALM protocols. Multicasting in CAN and in Chord using the forwarder driven approach does not support these node to root RTT constraints. Multicasting in NICE and Chord using the receiver driven approach can be extended to support these constraints.

OM-QoS was implemented as a framework for OMNet++. Therefore, other protocols can be implemented using our framework for OMNet++ to evaluate their QoS support. The OM-QoS framework and implemented protocols could also be adapted to be used in the Internet with real hosts rather than in a simulator. Only minor changes would be necessary in order to achieve this transition from simulator to real networks environments.

In Chapter 4, we present a solution on how to bridge IP Multicast with ALM, which then can be used to enable also QoS for IP Multicast using OM-QoS. The next step would be to actually have an application, which enables end users to share files among themselves using multicasting paradigm (be it IP Multicast or ALM) and to additionally benefit from QoS mechanisms. Therefore, we introduce MCFTP (Multicast File Transfer Protocol) in Chapter 5. It allows end users to exchange data efficiently like using BitTorrent but also facilitates using resources more efficiently due to multicasting mechanisms. We evaluated MCFTP using IP Multicast and ALM, where we also were able to setup the P2P/ALM network to support QoS regarding bandwidth requirements of nodes.

Chapter 4

Providing IP Multicast Services via P2P / Overlay Networks

4.1 Introduction

In this Chapter we describe the Multicast Middleware, a bridge between IP Multicast and Overlay Multicast, as presented in [114, 42, 31, 32, 40, 44, 19, 122, 38, 39]. It has been developed in the scope of the Sixth European Framework Program Integrated Project EuQoS (see Section 2.7.2). It facilitates using IP Multicast functionality by end users through the Internet without requiring any IP Multicast support on the intermediate routers. This is achieved by capturing IP Multicast packets on end systems and tunneling them through an Overlay Network. The whole process of capturing, tunneling and injecting the IP packets back into the system at the receiver's side is completely transparent to users and applications.

IP Multicast (presented in Section 2.2.2) is an efficient way to disseminate data from a sender to multiple receivers concurrently. Unfortunately, IP Multicast has not been widely deployed in the Internet today, although several applications exist that can use IP Multicast to transmit and receive data. To overcome this limitation, Overlay Multicast, also called Application Layer Multicast (ALM) presented in Section 2.5, which runs on-top of Peer-to-Peer (P2P) Overlay Networks presented in Section 2.3, has been introduced. End systems exchange the data directly among themselves using only unicast communication. They do not rely on routers in the network multicasting the data. Therefore, ALM can be used as an efficient data dissemination scheme/mechanism for video broadcasting, IPTV, multiplayer games, and other scenarios. Unfortunately, ALM is not a standardized protocol, such as IP Multicast, and hence different ALM protocols exist.

To make the use of ALM transparent to applications, we provide an IP Multicast service to end system applications by capturing the outgoing IP Multicast traffic from hosts and routing it using an ALM Overlay Network. This unique feature allows users on end systems in the Internet to benefit from easy deployment of P2P based ALM mechanisms and still to use existing IP Multicast enabled applications. To provide a platform independent solution, we used Java [80].

In Section 4.2, we describe the architecture and design of the Multicast Middleware. The implementation of the Multicast Middleware is illustrated in Section 4.3. The evaluation scenarios for the Multicast Middleware are presented in Section 4.4. Section 4.5 presents the evaluation results. Section 4.6 concludes this chapter.

4.2 Architecture and Design of the Multicast Middleware

4.2.1 Overview

The Multicast Middleware can be used with almost any ALM protocol that offers standard multicast operations (subscription to multicast groups, receiving and sending multicast data). Typical P2P / ALM networks try to approximate the efficiency of IP Multicast communication regarding link stress by using unicast communication. As shown in Figure 2.2, Application Layer Multicast is not able to totally avoid sending redundant data over the same physical link as IP Multicast does. However, it can significantly reduce the number of redundant data flows in the whole network. Overlay Networks are usually built in a topology aware manner. Therefore, peers, which are "close" to each other in terms of communication latency, are directly connected. The P2P links are constantly monitored, which enables reacting to failures in network communication or to failures of neighbor peers.

We have chosen to use the Scribe Overlay Multicast facility (presented in Section 2.5.3), which runs on top of the the Distributed Hash Table (DHT) P2P Pastry (presented in Section 2.3.3). Scribe / Pastry scales well for a large number of participants and multicast groups. A Java implementation in the form of Freepastry [75] exists. The Multicast Middleware is not limited to use Scribe / Pastry, but could also be adapted to use other P2P Overlay Multicast systems such as Chord (presented in Section 2.3.6), Tapestry (presented in Section 2.3.4), etc. For the Multicast Middleware implementation, we use one dedicated Scribe / Pastry Overlay Network instance per active IP Multicast group. To find the corresponding Scribe /Pastry instance, we also use one only for this purpose dedicated Pastry network to store the management information about active groups, such as the IP Multicast address, a number of hosts that are active in the Overlay Network, etc.

Eliminating the requirement for multicast support by the operating system and the underlying network makes the use of ALM very appealing for any kind of Internet users. The disadvantage of ALM is the lack of standardization – each ALM has its own API and addressing scheme. This prohibits already existing multicast-aware applications from using ALM. The Multicast Middleware uses ALM for transporting multicast traffic. However, it also offers a standard IP Multicast interface for the applications. This process is completely transparent to applications.

In order to support as many operating systems as possible, the core of the Multicast Middleware has been implemented using the JAVA programming language. The only non-portable component of the Multicast Middleware prototype is the communication/capturing module for the TAP interface, which will be explained

in Section 4.2.2. This module differs for each operating system and is usually implemented using C programming language.

Dr. Multicast (presented in Section 2.5.12) targets mainly data centers. Our approach aims to enable Internet-wide IP Multicast support. Furthermore, our solution supports all major operating systems (Win32, Mac OS X, Linux) while Dr. Multicast is limited to Linux. Also, network administrators do not need to configure or enable anything in their network in order to offer Internet-wide IP Multicast support. We rely solely on end systems and end users without requiring a designated centralized instance for control and management. Our solution is targeted at scalable and distributed scenarios. Furthermore, using our solution, users can define QoS requirements for the different IP Multicast groups they are interested in. QoS reservations are then performed in the underlying QoS-enabled network and the ALM distribution tree is set up to support QoS as described in Section 3.2.2.

4.2.2 Providing an IP Multicast Interface for Standard Applications

The IP Multicast interface for applications is usually offered by the operating system via sockets. Signaling from the operating system to a multicast-enabled router in the local network is performed using IGMP [60, 72, 48]. Sending IP Multicast traffic is not really different than sending IP unicast traffic. There are just a few differences. IP Multicast uses a reserved IP source/address range for multicast groups (groups of multicast receivers with one IP address per multicast group). Group members have to join and leave the multicast groups they are interested in. Also, only the UDP protocol is used with IP Multicast. On the link layer, multicast traffic is though handled differently. In Ethernet, IP packets with a multicast group as a destination address are mapped to an Ethernet multicast address.

The Multicast Middleware can be used for many IP Multicast applications and scenarios. For example, a video streaming service for a very large group of Internet users can be provided, without the need for large investments in infrastructure. Only a few requirements for sender and clients must be fulfilled. The sender of a video stream must have the Multicast Middleware installed on a computer, which is connected to the Internet. The Internet connection should support at least the bandwidth for sending the video stream once. Any application supporting streaming using IP Multicast can be used (for example VLC). Each client that wants to receive a video stream must of course also install the Multicast Middleware on his computer and must have Internet access. Any video application with IP Multicast support (like VLC) can be used for receiving the video stream.

The Session Announcement Protocol (SAP) [84] can be used to announce running or scheduled video broadcasts over IP Multicast. The SAP announcements include Session Description Protocol (SDP) [83] stream descriptions encapsulated in UDP packets, which are sent to a predefined IP Multicast group (for example IPv4 global scope session announcements are sent to 224.2.127.254) and port (9875). Since the Multicast Middleware enables IP Multicast on end systems, SAP can be used for announcing video broadcast transmissions.

Virtual Network Interface to Capture IP Multicast Traffic

To provide an IP Multicast interface for the whole system (including services integrated in the operating system's kernel), we propose to use a virtual Ethernet device (also known as TAP device [164] – a software analogy of a wiretap). The TAP interface is a special kind of network interface, which is seen by the operating system as a normal Ethernet device. However, instead of forwarding the Ethernet frames to a hardware device, the TAP interface forwards the received Ethernet frames to a user-space process. On the other side, the TAP interface forwards all Ethernet frames received from the user-space process as incoming frames to the operating system's kernel. TAP support exists for all major operating systems such as UNIX/Linux, Mac OS X and WIN32.

Using a TAP interface together with the Multicast Middleware makes processing of multicast traffic transparent to all applications. This includes the multicast functionality integrated into the operating system's kernel. This approach does not require any modification of application code. Multicast traffic originating from an end user host can be routed through the TAP device. This device forwards the packets (encapsulated in Ethernet frames) to a user-space process (the Multicast Middleware) for processing. The Multicast Middleware acts like a multicast router by implementing IGMP and transporting the multicast data using ALM. Furthermore, the Multicast Middleware is able to send IP Multicast traffic back to the end system through the same TAP device.

By setting appropriate routing table entries for IP Multicast addresses on end systems, those packets are directed to the virtual network interface instead of the real physical network interface. All IP Multicast traffic will be redirected to the Multicast Middleware entity running on the end system. By doing so, the Multicast Middleware is aware of every IP Multicast group, to which the end system is subscribed to.

Applications on an end system with a running Multicast Middleware use the standard IP Multicast group management system calls. IP Multicast enabled applications must subscribe to different multicast groups to receive for example video broadcast announcements and audio/video streams. The multicast group subscription is usually a system call, which instructs the operating system's kernel to send IGMP membership report messages to the IP Multicast router in the network.

The operating system communicates with our Multicast Middleware through the virtual network interface and views it as an IP Multicast router. The Multicast Middleware interprets the IGMP membership reports and notifies the neighbor peers in the ALM Overlay Network about the changes in the multicast routing table. This information (depending on the multicast routing protocol used in the Overlay Network) is propagated to other peers. Every IP Multicast packet that should leave the end system is forwarded to our Multicast Middleware through the virtual network interface and then forwarded using the Overlay Network according to the routing setup in the ALM.

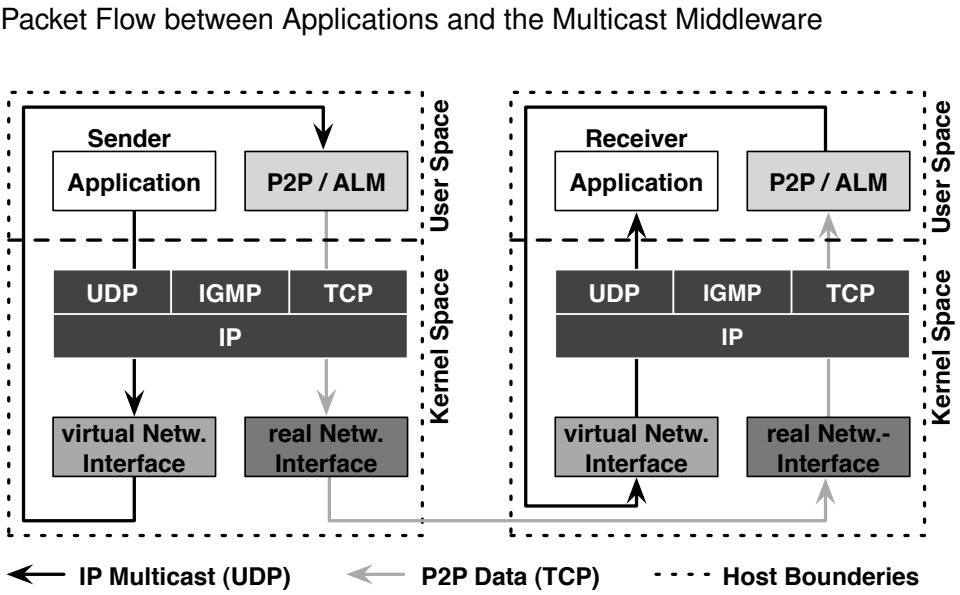


Figure 4.1: Packet Flow between Applications and the Multicast Middleware

Figure 4.1 shows how applications send and receive IP Multicast traffic using the Multicast Middleware. The virtual network interface TAP captures/injects IP Multicast packets at end systems and then passes/receives IP Multicast data to/from the Multicast Middleware, which disseminates the packets among the group subscribers using ALM.

In detail this means that after a IP Multicast data packet for a specific IP Multicast group has been sent by the application, it is forwarded by the operating system's kernel to the appropriate multicast-enabled network device (in our case the TAP device). The Multicast Middleware process receives the outgoing IP Multicast packet via the TAP device. This packet is then encapsulated into an ALM message. The specific IP Multicast destination group address of the packet is translated into an ALM address to which the ALM message is sent. After receiving the ALM message with the encapsulated IP Multicast packet, the Multicast Middleware extracts the IP Multicast packet from the ALM message and encapsulates the IP Multicast packet into an Ethernet frame. The Multicast Middleware then sends the Ethernet frame via the TAP interface to the operating system's kernel for processing. The operating system's kernel delivers the IP Multicast packet to the application that subscribed to that specific IP Multicast address.

4.2.3 Mapping IP Multicast Addresses and Messages to ALM

For transporting multicast traffic we do not propose a new P2P/ALM protocol. Almost any ALM protocol can be used and integrated. In the following paragraph, we describe the requirements for the ALM protocol and how IP Multicast traffic can be mapped to ALM messages.

Every IP Multicast packet has either a source or a destination address out of the IP Multicast address range (224.0.0.0 to 239.255.255.255). The IANA Guidelines for IPv4 Multicast Address Assignments [6] define how addresses from this range should be used and assigned.

Most ALM protocols implement their own multicast addressing scheme. Depending on the protocol's addressing scheme, the address range can be smaller, equal or larger than the IP Multicast address range. In case of a larger or equal address ranges, multicast addresses can be mapped 1-to-1 to the ALM addresses.

For example, the IP Multicast address range can be mapped to a consecutive address range of the same size in the ALM protocol's addressing scheme. In the case where the address range of the ALM is smaller than the IP Multicast address range, the IP Multicast addresses must be projected to the ALM address range. This can be achieved by hashing the modulo-function: $A_x = (A_{\text{IP}} - 0xe0000000) \bmod \#$ of addresses. IP Packets can be encapsulated in ALM messages. If the maximal length of an ALM message is lower than the IP packet's length, standard IP packet fragmentation can be applied to the packet in order to transport the packet through the Overlay Network. On reception of fragmented IP Packets, the Multicast Middleware should be able to de-fragment them and to deliver them to the TAP interface. The time to live (TTL) field of transported packets should be reduced for each P2P hop. Packets with TTL 0 should not be forwarded.

4.2.4 Security and Privacy Considerations when using ALM

In contrast to IP Multicast, where routers in the transporting network replicate data packets, ALM relies on end systems to replicate data packets. As a consequence, not only ISPs have the possibility to monitor the traffic, but also end users can "see" the traffic, which their neighbors are receiving. Hence, the security and privacy of the end users are even more threatened than with IP Multicast.

End systems, which are used as relays for multicast data can accumulate knowledge about the preferences of their neighbors regarding the reception of video streams. This information can be used for targeted marketing of products or as a component of user surveillance. This effect can be amplified through the collusion of relaying peers, which can exchange the surveillance data about neighbors. This facilitates creating a more complete picture of the monitored peers. Malicious peers can also alter the video stream they are relaying. For example, a malicious peer could inject commercials or logos into the video stream. Such behavior can be detected [82] and appropriate actions (e.g., excluding the peer from the overlay network) can be taken.

The privacy of end users can be improved by using ALM routing protocols, which change the delivery path of the multicast data over time or use parallel paths for receiving data. In this way, one peer is not always relaying data for the same peer and is not able to accumulate the information about involved end users.

Since peers, which are subscribed to one multicast group do not only receive multicast data for that group, relaying peers can also receive the multicast data they

are forwarding. To protect commercial content, some kind of content encryption has to be introduced. A possible content encryption and authentication solution is described in [14]. Another alternative is to construct an Overlay Network consisting only from receiving peers for each multicast tree, which is the solution we have chosen for the Multicast Middleware when using Scribe / Pastry. The disadvantage of this approach is the higher number of Overlay Networks, in which one receiver is participating, if he wishes to receive more than one video stream.

We also proposed REPOM (Reputation Based Overlay Multicast) [47], which helps to identify non-cooperative or selfish nodes in a distributed manner. Nodes gossip [66] reputation reports to other nodes in the P2P/ALM network. According to reports received, nodes can decide if they look for a better performing (non-selfish) parent node for multicast data subscription.

4.3 Implementation of the Multicast Middleware

4.3.1 Overview

The Multicast Middleware uses Pastry (see Section 2.3.3) as a P2P routing substrate and Scribe (see Section 2.5.3) to handle multicast group management and overlay handling. In the Multicast Middleware, one dedicated Pastry network is created for each active IP Multicast group. This ensures that only end systems, which are interested in receiving the multicast data of certain a group, are used to forward traffic. We implemented our own efficient P2P data forwarding protocol for data distribution and did not use the multicast data distribution mechanism of the Scribe / Pastry implementation Freepastry, which is not designed for high bandwidth scenarios. Finally, we implemented mechanisms to map IP Multicast protocol and data messages to the appropriate Scribe/Pastry network instances. We wanted to ensure that we could later easily replace the protocol used to handle multicast group management and overlay handling. Therefore, instead of improving the multicast data distribution method used in Freepastry, we implemented our own optimized P2P data distribution protocol. We separated the multicast data forwarding/distribution mechanism and protocol from the ALM group management and overlay handling protocol. Hence, Scribe/Pastry can be easily replaced by another P2P/ALM protocol, while still being able to use our optimized P2P data distribution protocol.

4.3.2 Using Freepastry for P2P/ALM Topology Management

Freepastry [75], an open source Java implementation of Scribe / Pastry, is used as the base implementation for the ALM. Freepastry relies on Java object serialization, which is not optimal for transporting data over the Internet. The reason for this is that each time a Pastry message is de-serialized a new instance of a Java object is created. If we would be using Pastry messages to transport the IP Multicast packets, each time one packet is received from the Pastry network at least

one Java object would be created. Due to automatic garbage collection mechanism in the Java virtual machine, these new objects would be de-allocated only when the heap of the Java virtual machine is full. Since the code execution of the Java virtual machine is paused during garbage collection, the packet delivery would be suspended and this would lead to increased packet delays or packet drops. We have therefore decided to use Scribe / Pastry only to construct the topology of the overlay network.

4.3.3 Efficient P2P Protocol for Multicast Data Transport

For transporting IP Multicast traffic, we create an Overlay Network with the same topology as constructed by Freepastry. But, we are using our own high performance optimized P2P data distribution protocol instead of relying on Freepastry's multicast data distribution protocol, which uses the object serialization mechanism of Java to process and forward the multicast data. This high performance optimized P2P protocol is based only on copying buffers without generating any Java objects. Such a design allows us to provide high performance multicast data transfer between end systems in the Internet, which is crucial for multimedia applications such as video streaming, video conferencing, etc.

We have implemented a simple P2P protocol that is highly efficient and only introduces minimal overhead in terms of processing required by the end system. The protocol uses TCP connections between the peers, through which the messages are exchanged. The message format resembles the format of messages used in the Common Open Policy Service (COPS) protocol [2]. The reason for using the COPS protocol message format is its extendability and efficiency. Each message consists of a header, which defines the length and type of the message and a list of objects, which contain additional information. The types and order of objects in a message depend on the type of the message. Each object consists of a header and data. The header defines the length, class and type of the object. The object data describes the object. Depending on the class and type, the semantic of the object data differs. For example, the object data can be an IP address or a encapsulated IP packet. This kind of design ensures easy protocol extension by introducing new types of messages or by adding new classes and types of objects into the existing messages.

Open Message The *Open* message is exchanged between the peers as soon as a P2P connection has been established. With the Open message each peer informs its communication partner about his preferences for receiving multicast traffic. The peer defines the UDP port on which the peer wishes to receive the multicast traffic encapsulated in UDP unicast packets. If the port is undefined, the peer can only receive multicast traffic through the already established P2P TCP connection.

Keep Alive Message The *Keep Alive* (KA) message is sent by each peer periodically to verify the TCP connection. If no Keep Alive message is received

within the predefined timeout period (10 seconds) the TCP connection is considered invalid and is closed.

Add Route Message The *Add Route* and *Remove Route* messages are used to signal to peers changes in the multicast routing table.

IP Data Message The *IP Data* message contains an encapsulated IP packet (for example a IP Multicast packet). This message is used for transporting multicast traffic through the P2P network.

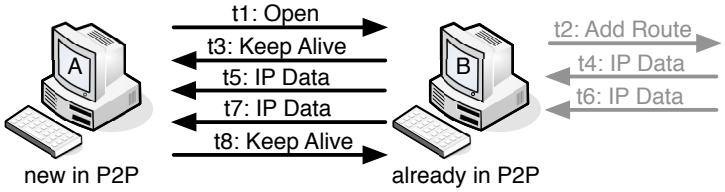


Figure 4.2: Example Message Exchange using our Simple P2P Protocol

An example of a message exchange using our simple P2P protocol is presented in Fig. 4.2. First, peer A, which is newly joining a P2P network sends an *Open* message to open the connection to peer B, which is already in the P2P network (at t1). Peer B then informs his other neighbor peers in the P2P network about the changes using an *Add Route* message (at t2). Then, peers need to keep the connection open by periodically sending *Keep Alive* messages (at t3 and t8). When encapsulated IP packets have to be forwarded through the P2P network, then they are transmitted using *IP Data messages*, first to peer B from its neighbor (at t4 and t6) and then from peer B to peer A (at t5 and t7 respectively).

4.3.4 Multicast Subscription Handling and Forwarding

The Multicast Middleware implements currently IGMP version 1 [60] for handling IP Multicast subscriptions. The Multicast Middleware sends periodically IGMP host membership query packets on the TAP interface. The operating system's kernel replies to each host membership query with one IGMP host membership report message for each IP Multicast group to which at least one application has subscribed. The removal of a host membership is not signaled explicitly in IGMPv1. Hosts simply do not send membership reports for the group to which they are not interested any more so that the membership for the group expires. As consequence, a certain lag between an application signaling the operating system to leave the group is introduced.

The Multicast Middleware forwards the multicast traffic between the local TAP device and other peers according to a multicast routing table. The multicast routing table does not distinguish between TAP interfaces and connections to other peers. Each entry in the multicast table consists of a multicast group and a set of "IP Multicast destinations" to which the packet is sent. The multicast table changes

through external events such as receiving add/remove route messages from connected peers. It also changes after receiving IP Multicast subscriptions from local applications through the TAP interface using the IGMP protocol.

4.4 Evaluation Scenarios for the Multicast Middleware

4.4.1 Overview

We evaluated the Multicast Middleware using different scenarios. First, we performed a functional test to validate and evaluate the functioning of our first prototype implementation. Then, we analyzed the throughput and loss performance of the Multicast Middleware using different chain topologies. Finally, we also performed delay and loss measurements for scenarios with chain and tree topologies. We will first present the different scenarios used for the evaluation in Sections 4.4.2 to 4.4.4.

4.4.2 Functional Test Evaluation Scenarios

For the functional test of the first prototype of the Multicast Middleware as presented in [114], we used a simple P2P network topology, which was setup statically with configuration files. The P2P structure had to be built as a cycle-free tree. Multicast group subscriptions were flooded through the P2P network for this first test. We tested the functioning our prototype implementation of the Multicast Middleware by streaming a Video CD (MPEG 1 [96] stream with 1.4 Mbps) using Video LAN Client (VLC) [174]. The test-bed consisted of four mixed performance laptop computers connected via 100 Mbps Fast Ethernet:

- Sony Vaio with a 2.6 GHz Pentium 4 CPU and 1024 MB RAM
- Apple PowerBook with a 1 GHz PowerPC G4 CPU and 1024 MB RAM
- Dell Laptop with a 1.2 GHz Pentium 3 CPU and 256 MB RAM
- Dell Laptop with a 700 MHz Pentium 3 CPU and 256 MB RAM

The PowerBook was running on Mac OS X 10.4 Tiger, the other laptops were running Fedora Core 3 with a Linux Kernel 2.6.9. All laptops used Java Version 5 and had the TUN/TAP kernel modules installed.

The overlay network and routing was configured using the previously mentioned P2P messaging system. The IP Multicast data was encapsulated in P2P messages and transported through the Overlay Network using TCP connections between peers. The structure of the P2P network between the involved peers is depicted in Fig. 4.3. The Vaio P4-2.6 GHz laptop was used as sender and all other laptops were receivers of the stream and displayed the video using VLC. The Dell laptop with 1.2 GHz had to replicate the received stream two times to send it to the remaining laptops (PowerBook 1 GHz and Dell P3-0.7 GHz). Note that the

Dell P3-1.2 GHz in the center of the distribution tree had the triple network and processing load compared the other three laptops. It had to receive the stream and then replicate it twice to serve the remaining peers in the network.

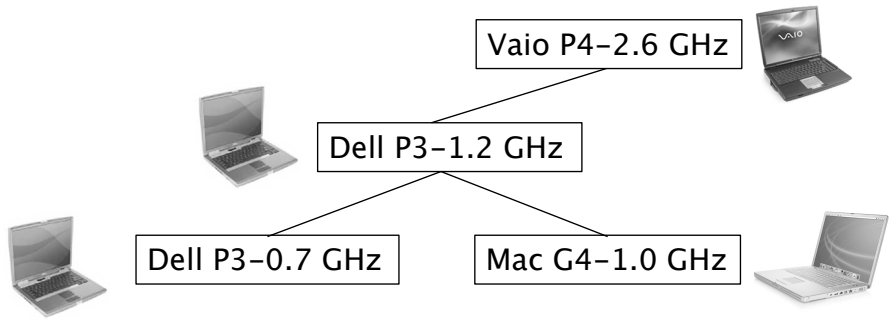


Figure 4.3: P2P Overlay Network Scenario for the Functional Test Evaluation

4.4.3 Throughput and Loss Evaluation Scenarios

Usually P2P network implementations like Scribe / Pastry based on Java or other high level programming languages are not very appropriate for high-bandwidth streaming. This is due to the fact that object serialization is normally used for sending P2P messages through the Overlay Network, as we also did for the first prototype of the Multicast Middleware. The object serialization implies usually a performance penalty due to the overhead of (de-)serializing objects. To overcome this problem, we developed a custom binary signaling protocol, and therefore can avoid creating new objects in most cases. This makes the Multicast Middleware then usable for high bandwidth and real-time critical environments.

To construct an Overlay Network, we use the mechanisms of Scribe / Pastry. But for the actual data transmission we are now using our optimized messaging protocol, which has been thoroughly optimized for high bandwidth data dissemination scenarios. We observed that by using the optimized communication protocol, the performance improved by one magnitude compared to the default P2P message protocol of Scribe / Pastry. Therefore, the following performance evaluation using MGEN [113] is focusing on determining the maximum achievable bandwidth that can be processed between two peers. Furthermore we also evaluate the performance of an example path from sender to receiver in a simple example Overlay Network.

Since we use a Scribe / Pastry Overlay Network structure, which is known to scale very well with an increasing number of peers and group members, we do not evaluate how well the Overlay Network performs in these terms. Due to limited bandwidth in distributed testbeds such as PlanetLab or the EuQoS testbed, it is not feasible to perform distributed performance measurements evaluating throughput and delay.

In order to evaluate the maximum sustainable bandwidth, which an instance of the Multicast Middleware can process, and to investigate the impact of forwarding IP Multicast traffic through ALM, we performed a series of tests with two different chain scenarios, which are shown in Fig. 4.4.

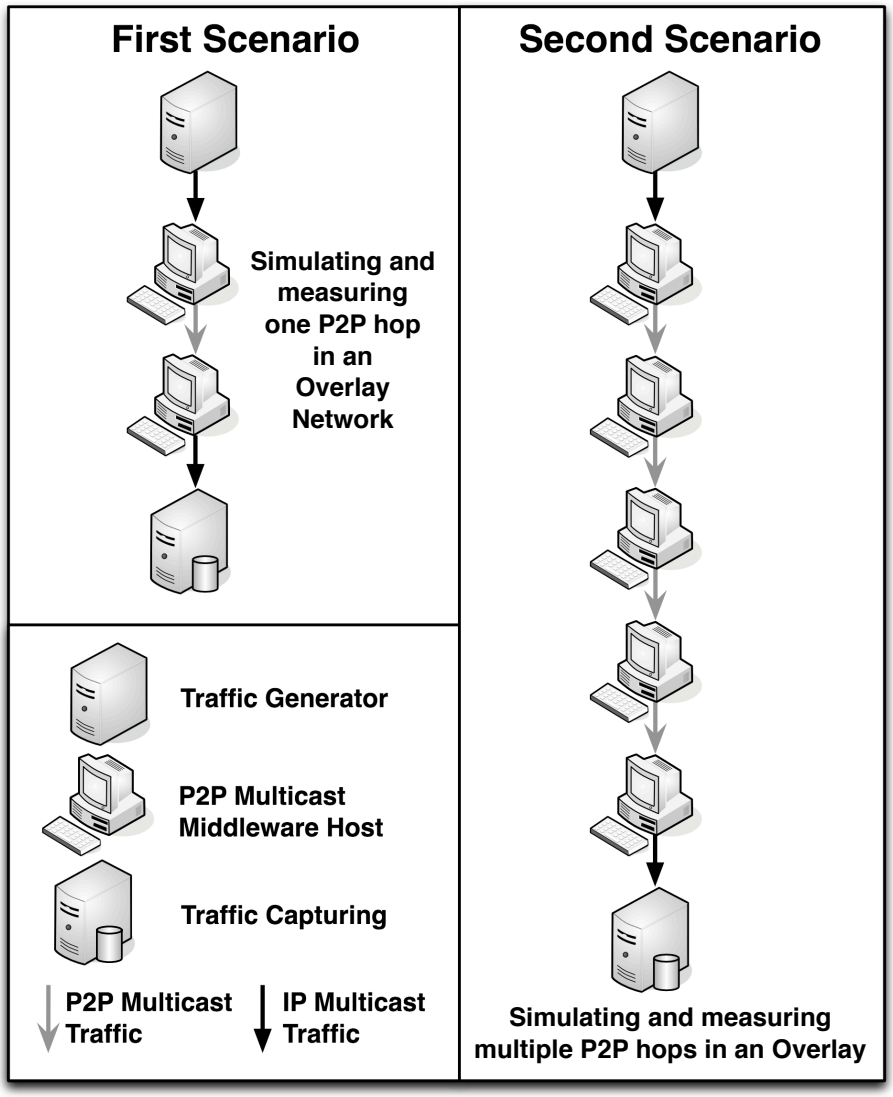


Figure 4.4: Scenarios for the Multicast Middleware Throughput and Loss Evaluation

The first scenario is a P2P network consisting of only two end systems. To avoid the interference of packet generation to the performance of the Multicast Middleware, we generated the traffic on separate hosts and forwarded it through a Gigabit Ethernet to the first peer. In the first peer, we used the Ethernet bridge functionality of the Linux kernel to interconnect the Ethernet interface with the virtual network interface (TAP). For the same reasons we built a similar scenario for cap-

turing the traffic. The goal of this setup is to determine the maximum throughput of the best case in the P2P network, where IP Multicast traffic is tunneled directly from a sender to a receiver.

To determine the effect of chaining multiple peers to forward the traffic, we designed the second scenario. In that scenario, we tunneled the IP Multicast traffic through five peers in a chain. This simulates an example of a sub path taken on one branch of the multicast tree in an Overlay Network. The first and last peer in this chain were connected to a traffic generator and capture point in the same manner as in the first scenario.

We used the MGEN [113] traffic-generating tool to generate and capture the IP Multicast traffic. The evaluated scenarios shared the following MGEN parameters and properties:

- 24 flows with sending rates ranging from 11 to 241 Mbps (steps of 10 Mbps)
- Flow sending duration of 120 seconds
- Packet payload of 1024 bytes

All PCs (traffic generator, traffic capturing and P2P Multicast Middleware hosts) in Fig. 4.4 were configured as follows:

- Intel Pentium D 3.0 GHz CPU with 2 MB Cache
- Two times 512 MB Take M5 DDR 400 CL 2.5 RAM
- ASUS P5LD2-VM mainboard with BIOS rev. 0606
- One Marvell 88E8053 PCI-E Gigabit onboard network adapter
- One Intel 82573L Gigabit LAN network card
- Hitachi Deskstar 7K80 80GB HDS728080PLAT20 hard drive

The operating system used was Fedora Core 5 with Linux kernel version 2.6.17. All hosts were interconnected via Gigabit Ethernet.

4.4.4 Delay and Loss Evaluation Scenarios

The delay and loss measurements were performed using SmartBits [158] “Portable High-density Network Performance Analysis System”. We compared the performance of native IP Multicast and IP Multicast tunneled through Overlay Multicast with the Multicast Middleware. We used the SmartBits to generate and capture the traffic for comparing latency and packet loss. These measurements focus on the achievable throughput and the latency on end systems with acceptable packet loss for high-bandwidth scenarios, such as video broadcasting or IPTV, and for real-time and interactive applications, such as VoIP and multiplayer games.

Topologies used for Measurements

We have conducted performance measurements with several chain and tree topologies. We will look at a chain topology consisting of four hosts and a tree topology with the maximum tree depth of four.

The multicast fan-out describes the number of nodes that a multicast parent has to serve with multicast data. Each host has a multicast fan-out of one in the chain scenario, so no IP Multicast packets need to be duplicated but just need to be forwarded. In the tree scenario, three hosts have to duplicate the data, and therefore having a multicast fan-out of two. The remaining four hosts in the tree scenario have a multicast fan-out of one.

The tree scenario comes close to an actual sub-part of a big overlay, where typically the multicast fan-out of hosts is between 0 to 2. We used traffic characteristics reflecting a constantly sending source at various fixed rates (1.0, 5.0, 10.0, 24.8, 49.6, 75.2 and 84.8 Mbps) with packet payload sizes of 32, 512 and 1024 Bytes.

The traffic was generated and measured using the SmartBits, which facilitates more accurate measurements than the software based solutions running on a Linux machine as performed in the previous measurements using MGEN. The Linux routers in-between were running no background processes, their full system performance was available for the IP Multicast routing in the native IP Multicast measurements, and for the Multicast Middleware packet capturing/processing and Overlay Network forwarding in the Overlay Multicast measurements with the Multicast Middleware.

The different topologies used for the performance measurements are depicted in Figures 4.5 and 4.6. The chain topology in Fig. 4.5 consists of the SmartBits (traffic generation and capture) and four PCs acting as Linux multicast routers connected in a chain.

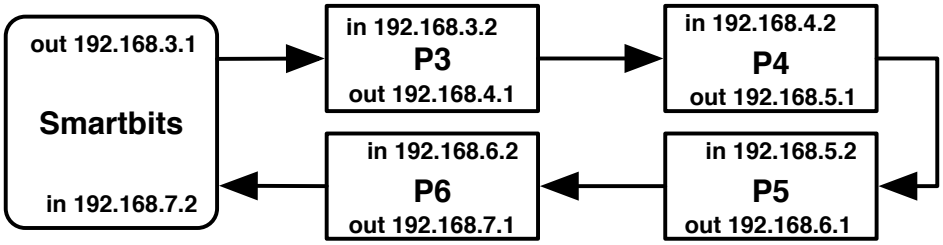


Figure 4.5: Chain Topology Scenario used for the Delay and Loss Evaluation

The tree topology in Fig. 4.6 consists of the SmartBits (traffic generation and capture) and seven PCs acting as Linux multicast routers. They are connected in such a way that the longest branch of the tree consists also of four PCs, but the branching nodes have to additionally duplicate the packets and send them to two receivers.

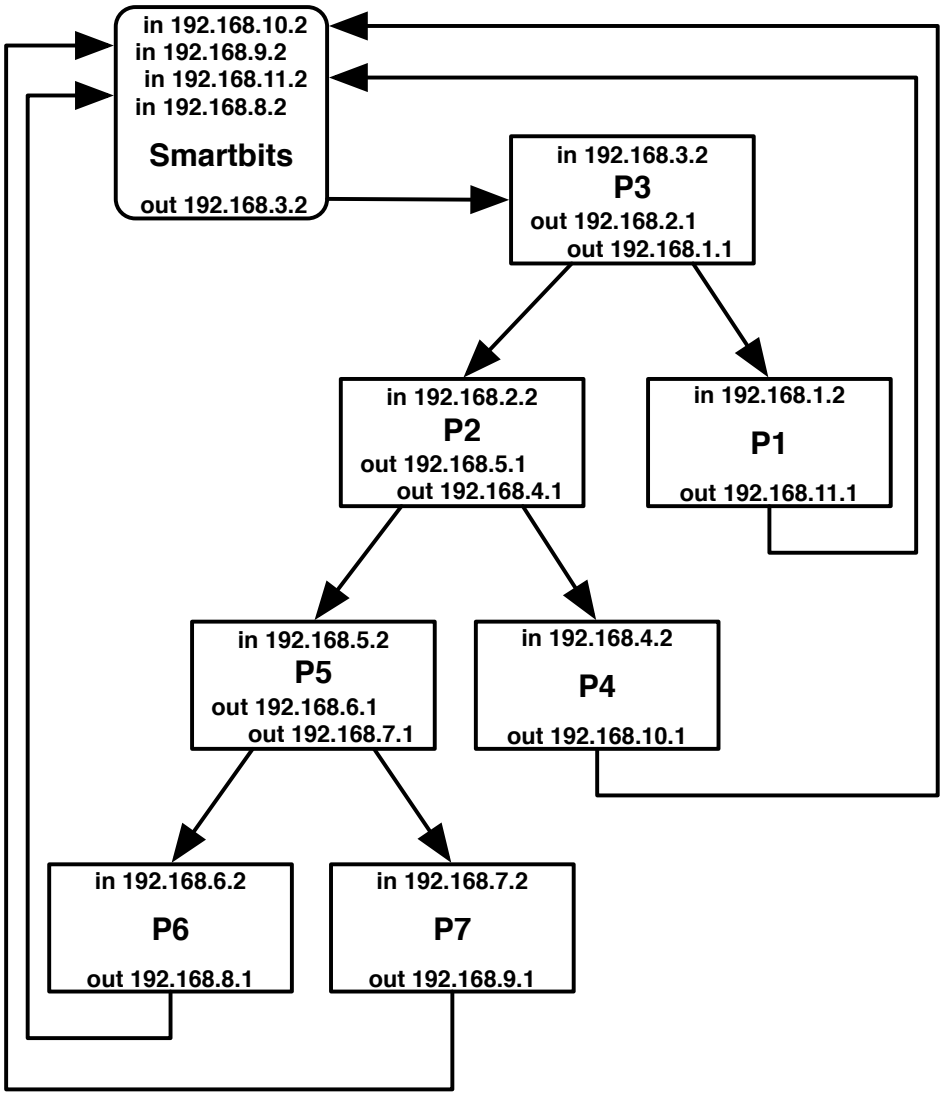


Figure 4.6: Tree Topology Scenario used for the Delay and Loss Evaluation

Systems and Environment used for the Measurements

We wanted to accurately measure and compare the latency introduced by processing and tunneling the IP Multicast data through Overlay Multicast with the Multicast Middleware and also wanted to determine the maximum achievable throughput, the packet loss and delays using the Multicast Middleware in different topologies. Unfortunately, there are no such large pure IP Multicast networks available in the Internet to support these high-bandwidth measurements. Additionally, the measurements to determine the introduced delay as described above would not be possible in the Internet. Therefore, we also could not use the PlanetLab [131, 129, 130]

environment, which would introduce additional overhead and delays through the virtual machines used on PlanetLab nodes. Hence, we performed the measurements locally in a fully controllable laboratory environment.
PCs (P1–P7 in Figures 4.5 and 4.6) acting as routers in the test environment were configured as follows:

- Intel Pentium IV 3.0 GHz CPU with 1 MB Cache
- Two times 512 MB Take M5 DDR 400 CL 2.5 RAM
- ASUS P4S800-MX mainboards with BIOS rev. 0501
- One SiS 900/7016 100 Mbps onboard network adapter
- Two Realtek RTL-8169 Gigabit LAN network cards
- Hitachi Deskstar 7K80 80GB HDS728080PLAT20 hard drive

The operating system used was Fedora Core 5 with Linux kernel 2.6.20-1.2307. All hosts were interconnected via Gigabit Ethernet. To configure IP Multicast routing on Linux, we used SMCRoute [150] version 0.92. SMCRoute is a combination of a daemon and command line tool, which facilitates setup of static IP Multicast routes for the different network interfaces of the Linux routers. It is similar to mrouted [71], with the difference that it supports static instead of dynamic multicast routes, which is what we needed for our experiment setup, since we wanted to maintain static multicast routes.

The throughput and delay measurements using MGEN with two and five host chain scenarios showed that the Multicast Middleware running on similar hardware is able to process roughly 100 Mbps of total network traffic (incoming, duplicating and outgoing) without any packet drops. In those previous measurements, we used a single Dual-Core Pentium D 3.2 GHz CPU system for generating, capturing and forwarding the data. Since two different hosts were used to generate and capture the traffic, delay measurements as presented in this Section were not possible. This is due to the fact that it is very hard to synchronize the clock accurately between two independent systems.

Using SmartBits allows us to perform delay specific measurements and also allows us to accurately generate traffic with smaller packet sizes at high bandwidth, which would not be possible with software based systems.

Performed Measurements

We defined three different packet payloads in order to analyze differences between native IP Multicast and the Overlay Multicast solution using the Multicast Middleware. For each of the different packet payloads, we performed the measurements with five different network load values. The nature of the “SmartBits” API restricted us to use the inter-packet gap instead of directly the bandwidth as a parameter.

The different payload sizes, packet inter-arrival delay, resulting bandwidth and number of packets being sent are shown in Table 4.1 with the following meaning:

Payload: the actual payload size in a data packet.

Data: the overall packet size (including IP header).

Packet gap: the time in microseconds between the moment a packet has been completely sent and when the next packet starts being sent.

 # packets: number of packets sent in a scenario is shown in the row

The total number of packets that the SmartBits can process is limited. Due to this limitation, we did not send more than 130000 packets in any of the scenarios. Therefore the last two scenarios with a payload of 32 bytes had a runtime of less than half as long as the other scenarios. The combination of *Data* and *Packet gap* defines the desired *bandwidth* for the scenarios.

We used different payload sizes of 32, 512 and 1024 bytes with bandwidth values of 1.0, 5.0, 10.0, 24.8, 49.6, 75.2 and 84.8 Mbps. A smaller packet size resulted in more packets being sent and processed. This allows us to determine the limit of successfully processable packets for the different topologies. For the payload size of 32 bytes, we limited the bandwidth to 49.6 Mbps.

Table 4.1: Traffic Characteristics for the Delay and Loss Evaluation Scenarios

No.	Payload (bytes)	Data (bytes)	Bandw. (Mbps)	Packet gap (μs)	# packets
1	32	79	1.0	61000	8000
2	32	79	5.0	11600	40000
3	32	79	10.0	5500	80000
4	32	79	24.8	1720	130000
5	32	79	49.6	485	130000
6	512	559	1.0	430000	1200
7	512	559	24.8	12500	28000
8	512	559	49.6	4000	56000
9	512	559	75.2	1100	85000
10	512	559	84.8	450	95000
11	1024	1071	1.0	830000	600
12	1024	1071	24.8	24000	15000
13	1024	1071	49.6	7700	29000
14	1024	1071	75.2	2250	44000
15	1024	1071	84.8	960	50000

4.5 Evaluation Results for the Multicast Middleware

4.5.1 Overview

In the following sections, we present the results for the evaluation of the Multicast Middleware. We first present the functionality test results in Section 4.5.2. The functionality test has been performed by streaming a full-length movie and by evaluating the quality perception. The throughput and loss evaluation results are shown in Section 4.5.3. This evaluation has been performed using Multi-Generator MGEN software. Finally, Section 4.5.4 presents the results of the delay and loss evaluation. The measurements for the delay and loss evaluation have been conducted with the SmartBits "Portable High-density Network Performance Analysis System".

4.5.2 Functional Test Evaluation Results

For the functionality test to validate and evaluate the functioning of the first prototype of the Multicast Middleware, we used the scenario presented in Section 4.4.2. We were able to stream a full-length movie (141 minutes) without serious quality penalties using Video LAN Client (VLC). The quality perception was evaluated subjectively by two persons watching the movie using VLC. The only (occasionally) issue perceived at the receiver of the video stream was caused by packet delivery jitter. The jitter was caused by the JAVA garbage collection mechanism. In the first prototype of the Multicast Middleware, we did not use our own efficient P2P protocol for Multicast Data Transport. Instead, we used Freepastry's built in object serialization and de-serialization mechanisms to transport the multicast data. Therefore, we were creating one Java object for each IP Multicast packet intercepted on the system via the TAP interface.

The gained insights from this first evaluation lead to the conclusion that we needed to replace this mechanism of having one Java object for each IP Multicast packet with a more scalable solution, in order to achieve higher performance and to avoid any severe impact caused by the Java garbage collector.

4.5.3 Throughput and Loss Evaluation Results

Packet Loss Evaluation

The packet loss for the different transmission rates in both scenarios is shown in Fig. 4.7. The reason for packet loss is due to the incapability of at least one peer to process traffic at the given rate, which happens when the maximum processing capacity of an end system at a given time is reached.

The packet loss for a bandwidth up to 100 Mbps is negligible for both scenarios. For transmission rates of more than 100 Mbps, packet loss increases significantly. The packet loss is less than 4% for a bandwidth up to 155 Mbps. Therefore,

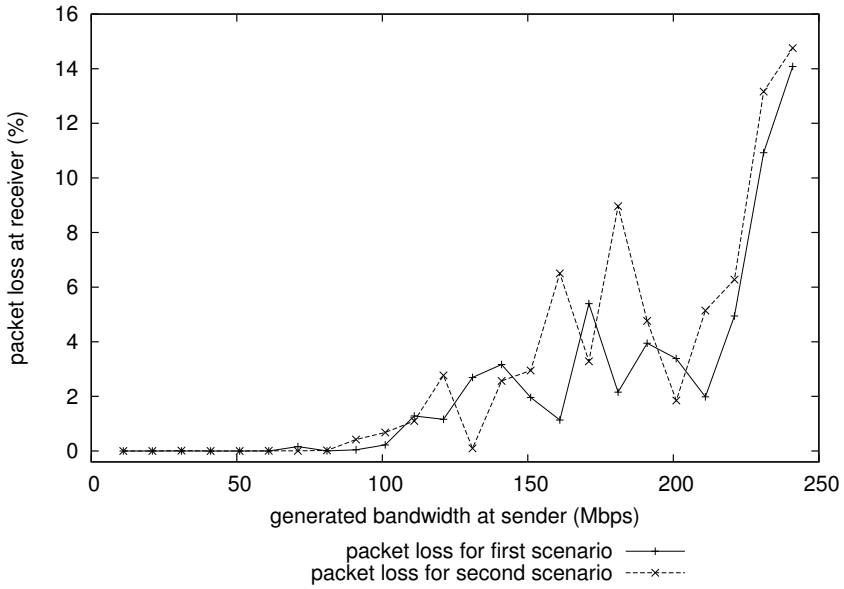


Figure 4.7: Packet Loss Measured for both Scenarios in Fig. 4.4

our Multicast Middleware should be able to support multimedia streaming up to 155 Mbps with acceptable packet loss.

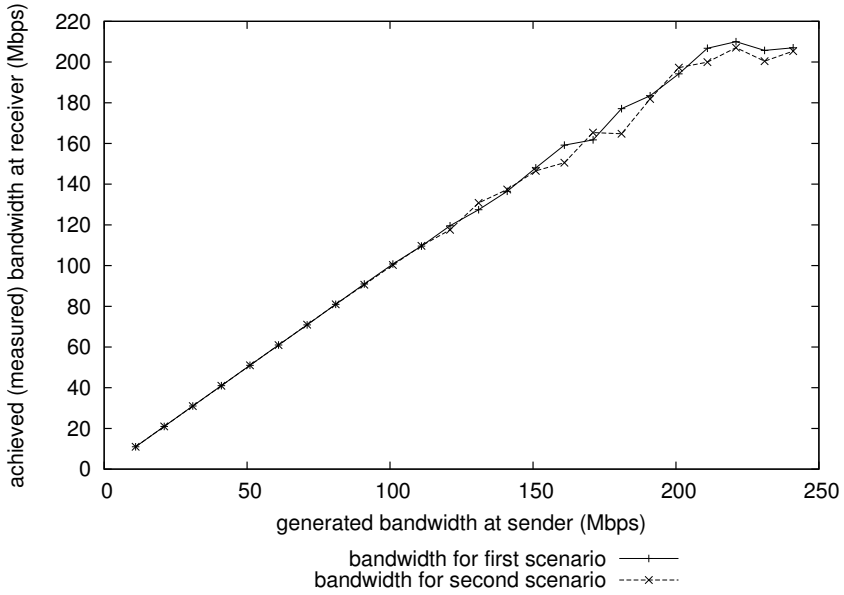


Figure 4.8: Generated and Effectively Achieved Bandwidth for both Scenarios in Fig 4.4

Bandwidth and Jitter Evaluations

Figure 4.8 shows the captured bandwidth compared to the generated bandwidth for both scenarios. It also shows that the maximum bandwidth an instance of the Multicast Middleware can deliver is at 210 Mbps.

We also determine the jitter for both scenarios. Our results show that the jitter increases with the number of peers involved in transporting the IP Multicast traffic. For the first scenario, the maximum jitter was below 15ms for a bandwidth up to 155 Mbps. For the second scenario, the maximum jitter went up to 150ms, due to a few outliers. The mean delay was much lower.

Both figures show that there is no significant difference between the packet losses for both scenarios, which indicates that the impact of delivering IP Multicast traffic through multiple peers is minimal.

4.5.4 Delay and Loss Evaluation Results

Native IP Multicast Results

The measurements with the chain topology in Fig. 4.9 show a very small packet loss below 0.2%. Most of the packets are lost in case of a packet payload of 32 bytes and higher bandwidth values. There is a high loss for high packet rates (small

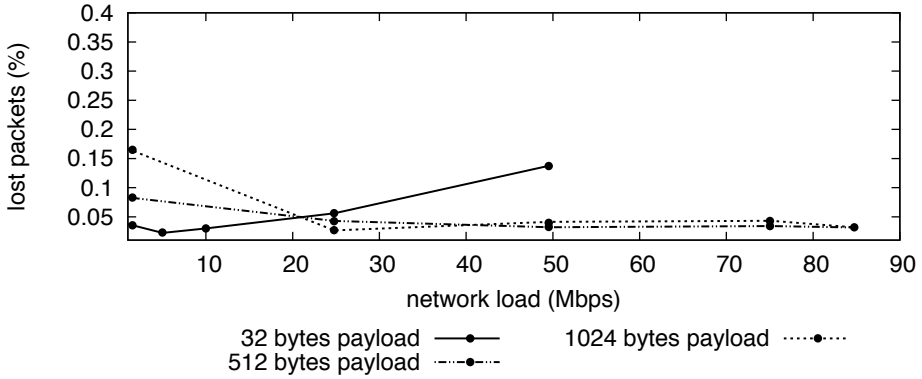


Figure 4.9: Packet Loss for IP Multicast in Chain Topology of Fig. 4.5

packet payload and high bandwidth). The measurements for a packet payload of 32 bytes have only been performed with bandwidth values up to 49.6 Mbps. For a payload of 1024 bytes, the packet loss rate starts just below 0.2% and then decreases towards 0% for higher bandwidth values. This is due to the fact that for low bandwidth values, only a few hundred packets are being sent and a single dropped packet has a significant impact on the percentage value, whereas for higher bandwidth values, many thousand packets are being sent, therefore reducing the impact of a single dropped packet on the percentage value. Higher bandwidth values are no problem for packet payloads of 512 and 1024 bytes.

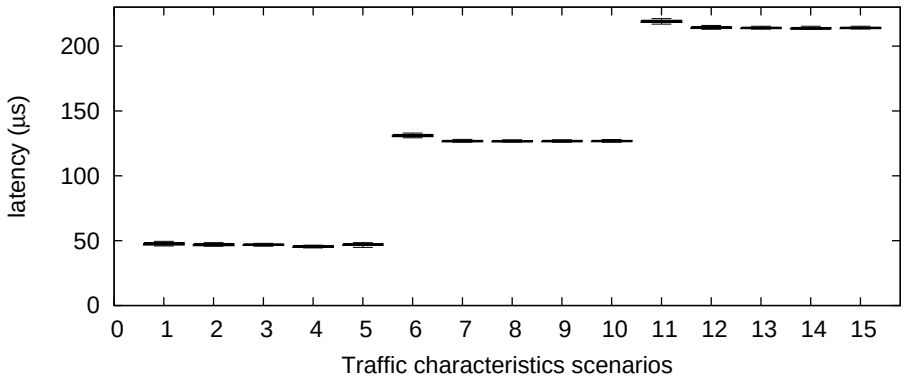


Figure 4.10: Latency w/o 5% Outliers for IP Multicast in Chain Topology of Fig. 4.5

The latencies for the chain topology are shown in Fig. 4.10. The network load has almost no influence on the latency, with negligible differences. It seems that the SMCRoute daemon (see Section 4.4.4) is buffering packets before forwarding packets to a host. Therefore, the delay increases as the payload size increases.

For the tree topology, we captured the data on the different lengths of the tree. The packet losses for P1, P4, P6 and P7 (leaf hosts) were similar, and therefore we only show the packet loss for P6 in Fig. 4.11. Compared to the chain topology,

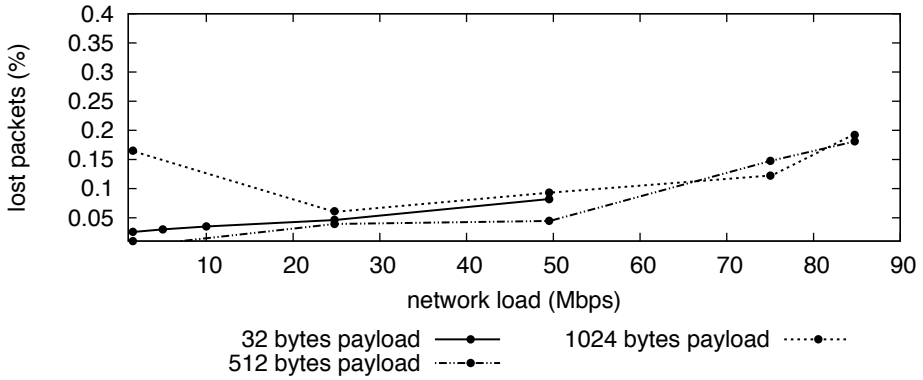


Figure 4.11: Packet Loss (P6) for IP Multicast in Tree Topology of Fig. 4.6

packet loss does not change significantly, although for configurations with higher payload, packet loss increases a little. The drop rate behavior for low bandwidth values and for a payload of 1024 bytes is as described in the chain scenario above.

The latency has a different behavior in the tree topology as shown in Fig. 4.12. It seems that SMCRoute does not buffer packets when it has to duplicate and forward them to multiple hosts. Hence, the average delay is now similar for different payload values. In the chain topology, the delay varied depending on the payload size due to the impact of buffering performed by SMCRoute.

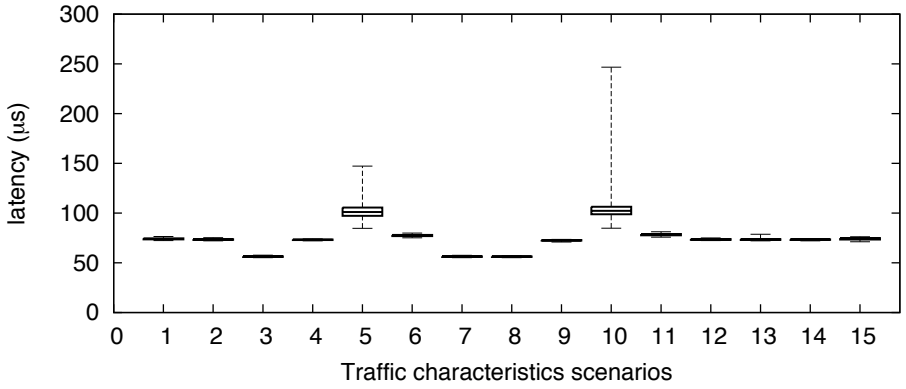


Figure 4.12: Latency w/o 5% Outliers (P6) for IP Multicast in Tree Topology of Fig. 4.6

We also have two configurations (traffic characteristics 5 and 10) with larger jitter than the average. It is clearly visible that the delay values measured have a much higher variance for those two configurations. For such configurations with a small payload and a high network load, the jitter seems to increase. This behavior is caused by the small inter-packet gap, and the amount of packets sent to the computers, as with this configuration, the kernel cannot transfer the packets quickly enough and they get queued.

Multicast Middleware Results

Our measurements show that the Multicast Middleware generally handles packet sizes of 512 and 1024 bytes very well while the smaller packet size of 32 bytes together with high bandwidth lead to a high packet drop rate. Figure 4.13 shows

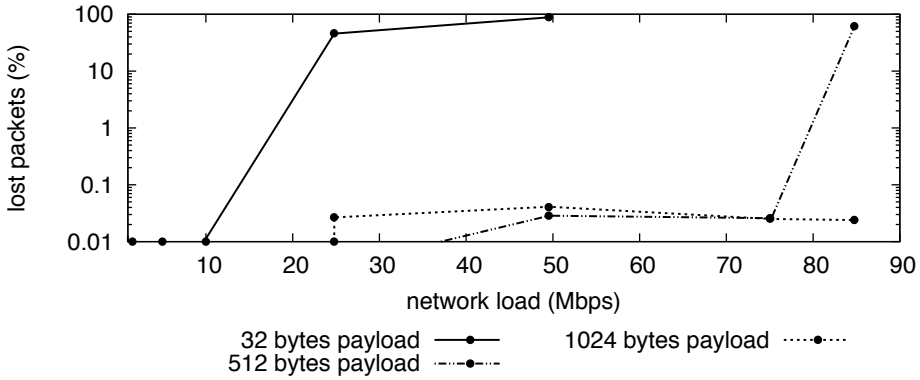


Figure 4.13: Packet Loss for the Multicast Middleware in Chain Topology of Fig. 4.5

that packet loss in the chain topology for 1024 bytes packet payload is between 0% and 0.04%. Packet drops occur when the Multicast Middleware has to process many packets in a short time, as it is the case for 32 bytes packet payload with

more than 10 Mbps and 512 bytes packet payload with 75.2 Mbps. Therefore, the Multicast Middleware seems to be able to process around 18'000 incoming packets per second in the chain scenario, before it starts to drop packets. The Multicast Middleware has to process two packet streams in the chain topology, the incoming and the outgoing stream.

The latency measurements for the chain topology using Overlay Multicast are presented in Fig. 4.14. The delays are acceptable (meaning below 10 ms) for band-

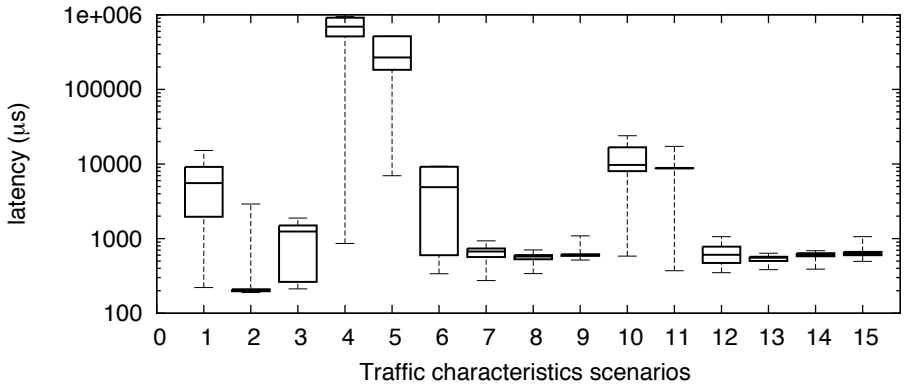


Figure 4.14: Latency w/o 5% Outliers for the Multicast Middleware in Chain Topology of Fig. 4.5

width values of 10 Mbps and below with a payload of 32 bytes. For 512 and 1024 bytes payloads, the latencies are acceptable for all bandwidth values used for the measurements. The latencies between 1 to 10 Mbps are decreasing for increasing bandwidth. This is due to the multi-threaded design of the Multicast Middleware.

Each overlay connection in the Multicast Middleware is handled by a separate thread (light-weight process). These separate threads read data from a ring buffer and wait on a conditional variable if their buffers are empty. At high packet rates, the threads never wait. At low packet rates though, the threads wait after transmitting each packet. Waking up threads produces additional latency in packet transmission.

Therefore, scenarios 1 to 3 result in higher latency for lower packet rates. The total number of waiting states per second is lower when the packet size is being increased. Therefore, the impact of multi-threading issues on the overall latency is reduced. This explains the smaller additional latency values in scenarios 6 to 8 and 11 to 13. The results obtained from the presented measurements in the chain topology using a packet payload of 1024 bytes correspond to the previously determined bandwidth limit of 100 Mbps with acceptable packet loss using measurements with MGEN.

The tree topology packet loss results presented in Fig. 4.15 show that packet loss is at an unacceptable level for 10 Mbps and more with 32 bytes packet payload, above 24.6 Mbps for 512 bytes packet payload and more than 49.6 Mbps for 1024 bytes packet payload. In the tree topology, the Multicast Middleware seems to be

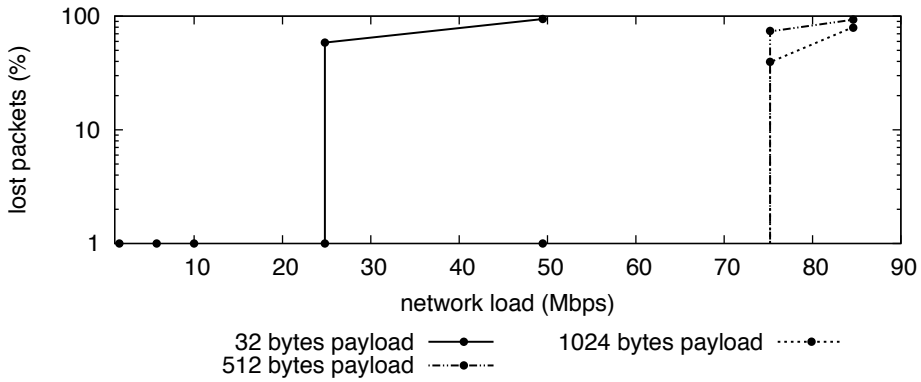


Figure 4.15: Packet Loss (P6) for the Multicast Middleware in Tree Topology of Fig. 4.6

able to process at least 9'000 incoming packets per second, before it starts to drop packets. In the tree topology, the Multicast Middleware has to process three packet streams, one incoming, and two outgoing.

The latency measurements presented in Fig. 4.16 lead to the same conclusion. The tree topology measurements show acceptable delays (below 10 ms) for band-

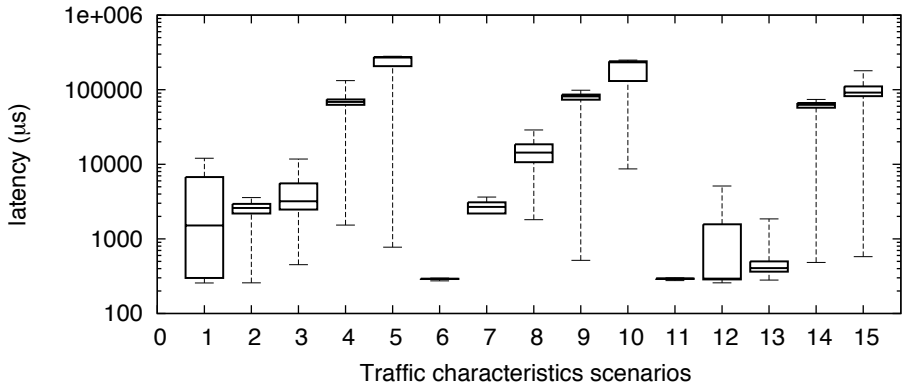


Figure 4.16: Latency w/o 5% Outliers (P6) for the Multicast Middleware in Tree Topology of Fig. 4.6

width values below 10 Mbps for 32 bytes, below 24.8 Mbps for 512 bytes and below 75.2 Mbps for a 1024 bytes packet payload. Our measurements show that the Multicast Middleware in the chain topology can process incoming packet flows up to 75.2 or 84.8 Mbps with a payload of 512 or 1024 bytes, respectively. The limit is above 10 Mbps in the chain topology for packet payloads of 32 bytes. For the tree topology, incoming flows up to 10 Mbps for 32 bytes payload and up to 24.8 Mbps for a payload of 512 bytes can be processed, while for a packet payload of 1024 bytes, incoming flows up to 49.6 Mbps can be handled.

These results show that the Multicast Middleware can be used for real-time

and high-bandwidth scenarios and that processing the packets introduces acceptable additional delays. The Overlay Multicast latency measurements results, which were still acceptable, are an order of a magnitude higher than with native IP Multicast. But average delays around 0.7 ms are still acceptable for the four hop chain scenario. In the tree topology, the average still usable delays are around 3 ms, which is still acceptable.

Comparison between IP Multicast and the Multicast Middleware

The loss rate in the chain topology of Fig. 4.5 does not behave very differently between native IP Multicast and IP Multicast tunneled through an Overlay Network using the Multicast Middleware for a packet payload size of 1024 bytes. Also for a packet payload size of 512 bytes, the values are in the same area for the different bandwidth values up to 75.2 Mbps. Above this bandwidth value, the Overlay Multicast scheme starts to have high losses.

For the small packet payload of 32 bytes, both scenarios behave quite similar up to a bandwidth of 10 Mbps, then the native IP Multicast scenario gets only slightly worse with raising bandwidth, whereas the Overlay Multicast scenario has high losses with bandwidth values above 10 Mbps.

Delays in the Overlay Multicast scenarios are much higher than for native IP Multicast, due to capturing, processing, replication and injection of packets using the Multicast Middleware and the Overlay Network. But we can see that this approach is valid because the introduced delay for most of the bandwidth and packet size scenarios is tolerable.

In the tree topology of Fig. 4.5, Overlay Multicast compared to native IP Multicast behaves almost the same, having the bandwidth limit for Overlay Multicast below 24.8 Mbps for packet payload size of 32 bytes and below 75.2 Mbps for payload sizes of 512 and 1024 bytes. The impact of capturing, processing, replicating and injecting packets using the Multicast Middleware on the delays in the tree scenarios is the same as for the chain scenario. Compared to native IP Multicast, the delays are higher, but behave less steady and increase if the bandwidth is augmented. But, the delays are still tolerable as described before for many packet payload size and bandwidth combinations.

Summary of Delay and Loss Evaluation Results

Using the SmartBits “Port-able High-density Network Performance Analysis System”, we were able to perform delay and loss measurements, which would not have been possible using software based systems such as MGEN. The results from the evaluation show that the Multicast Middleware can be used for high-bandwidth and delay sensitive scenarios, such as real-time video broadcasting, IPTV, multiplayer online games, NVE, and others. It is a valid solution to tunnel IP Multicast traffic through an Overlay Multicast network, in order to offer an Internet-wide IP Multicast service to end users.

4.6 Conclusion

IP Multicast is a necessary service for efficient video streaming in the Internet. Although commercial ISPs are not willing to enable IP Multicast services for end users, it is possible to provide multicast services using only unicast communication.

In this Chapter we described a way to provide IP Multicast services on end systems without changing existing IP Multicast applications or deploying infrastructure to the Internet. This facilitates efficient dissemination of multimedia data for video broadcasting, IPTV, VoIP and multiplayer games using the multicast paradigm. Our approach uses a virtual network device for capturing multicast traffic and forwarding it to a user application called the Multicast Middleware. The Multicast Middleware transports the multicast data using Application Layer Multicast to other receivers in the Internet. We have also described how existing Application Layer Multicast protocols can be integrated into our approach and outlined the possibility of QoS provision for the Multicast Middleware over the Internet.

The Multicast Middleware has been tested and evaluated using multiple scenarios and testing approaches. We compared native IP Multicast and Overlay Multicast with the Multicast Middleware regarding throughput, loss and delay performance. We concentrated our measurements on local networks to analyze the maximum throughput and the delays introduced by processing the packets, which would not have been possible in a distributed network over the Internet on a larger scale. We performed the measurements in different topologies (chain and tree) with variable network load and payload.

The results from the throughput, delay and loss evaluations show that the Multicast Middleware can be used for different scenarios. It supports high-bandwidth and delay sensitive scenarios, such as real-time video broadcasting, IPTV services, massive multiplayer online games, networked virtual environments, and others.

The Multicast Middleware has been developed in the context of the EuQoS project that aims to provide a facility for providing inter-domain end-to-end QoS services. The Multicast Middleware can use reservation-based DiffServ-like systems to perform QoS reservations on the network level for the connections between end systems. As an alternative, also best-effort measurement based QoS with measurements and/or predictions [148] could be used. Since the QoS requirements of the end systems within one IP Multicast group can be heterogeneous, it is necessary that the multicast tree is built in such a way that the QoS requirements and capabilities of end systems are considered. This will enable QoS not only for the reception but also the redistribution of the received encapsulated multicast data.

The Multicast Middleware has been successfully tested with legacy IP Multicast applications, such as Video LAN Client (VLC) [174] and software developed in student projects [19, 122]. It has been presented using VLC at conference demonstrations [44], EUQoS [68] project yearly reviews and roadshows [74].

In Chapter 3 we described the properties of a QoS aware multicast tree and how such trees can be constructed using Scribe / Pastry. The presented extensions to Scribe / Pastry have been integrated into the Multicast Middleware.

Chapter 5

Efficient Data Dissemination using Cooperation

5.1 Introduction

Efficient data distribution in the Internet can be performed using multicast mechanisms such as IP Multicast or Application Layer Multicast (ALM) using Overlay Networks. In this Chapter, we present the Multicast File Transfer Protocol (MCFTP) [45, 36, 37, 32, 33, 128, 77], which uses multicasting to distribute data efficiently to end users with varying network speeds.

Different approaches to efficiently distributing data among multiple downloaders / subscribers exist. One of the commonly used protocols is BitTorrent. It allows end users to share their resources in order to improve their download speed. BitTorrent is based on unicast connections and does not use or benefit from multicast.

We have developed MCFTP, a Multicast File Transfer Protocol, which performs efficient data distribution using multicast. Instead of IP Multicast (see Section 2.2.2), Application Layer Multicast (shown in Section 2.5), running on-top of Peer-to-Peer (presented in Section 2.3) Overlay Networks can be deployed for multicast to end users.

MCFTP incorporates some ideas of BitTorrent. Users that have downloaded partial data can contribute by uploading those already received parts to others. We compared MCFTP with BitTorrent in IP Multicast networks and using Overlay Multicast environments using the ns2 [70, 29, 65, 92, 119] network simulator.

Using multicast instead of unicast facilitates using resources more efficiently. This is the case for IP Multicast, where routers replicate data, as well as for ALM, where participating nodes are arranged to use their resources as efficiently as possible.

In Section 5.2, the architecture and design of the MCFTP protocol and components are described. The evaluation scenarios of MCFTP are presented in Section 5.3. The results of the evaluation, comparing MCFTP to BitTorrent using the ns2 simulator and a prototype implementation are presented in Section 5.4. Finally, Section 5.5 concludes this chapter.

5.2 Architecture and Design of MCFTP

5.2.1 Overview

Often, P2P protocols for data dissemination, such as Slurpie, BitTorrent, etc. as described in Section 2.3, use cooperation among the download clients to provide partially downloaded files to other downloaders. They all rely on 1:1 connections between clients and do not benefit from the multicast distribution paradigm. The Multicast File Transfer Protocol (MCFTP) offers a file sharing and data dissemination service based on multicast.

MCFTP has five main entities: the FileDescriptor, the FileManagementGroup (FMG), SendingGroups (SG), downloading nodes, and optionally a FileLeader. The FileDescriptor stores meta data (unique file identifier, file size, chunk size, hash values for chunks) about the file that should be downloaded / disseminated. Protocol related communication among nodes is done via the FMG. A FMG is a dedicated multicast group for a specific file. The application data itself (the file) is disseminated or transferred using dedicated multicast groups for file dissemination (SendingGroups), which are only available for a short life time.

Finally, MCFTP can run in a completely decentralized mode (called dMCFTP) or in a centralized mode. The centralized mode (called cMCFTP) has a FileLeader coordinating the SendingGroups.

5.2.2 FileDescriptors

Files that can be downloaded using MCFTP are described by a FileDescriptor. It contains all meta data and is a normal text file. A FileDescriptor consists of a unique file identifier, file size, chunk size and for each chunk a list of hash values. Files are split into chunks of equal size. The chunks are the smallest transmitted entities between nodes.

5.2.3 FileManagementGroup

The FileManagementGroup (FMG) is a multicast group, which is used to exchange status and coordination messages. It is a dedicated multicast group, which is unique for a file as described in the FileDescriptor.

In the centralized mode cMCFTP with a FileLeader, the FMG is used by the nodes to send status update messages to the FileLeader. Status update messages contain information about the parts or chunks of a file that nodes already possess. Additionally, status update messages also hold other information, such as a node's pre-configured upload and download bandwidth. Users can define the upload and download bandwidth that the application can use for itself. The FileLeader on the other hand uses this group to send coordination messages to all nodes. It announces which nodes need to send which chunks. The announcements are done via keep-alive messages and contain additional information about the SendingGroup and the sending rate.

Figure 5.1 shows a cMCFTP scenario with a FileLeader (FL). First, nodes (H1-H9) send their status update messages using the FMG. The FileLeader (H1=FL) then calculates availability and demand of the chunks. Therefore, the FileLeader can determine which nodes are the best senders (S1-S3) for the nodes (R1-R6) still missing some chunks. Only one node (S3/R5) acts as sender and receiver at the same time. It then announces via the FMG which nodes (S1=H2, S2=H5, S3=H8) have to send what chunks with which transmission rate how many times.

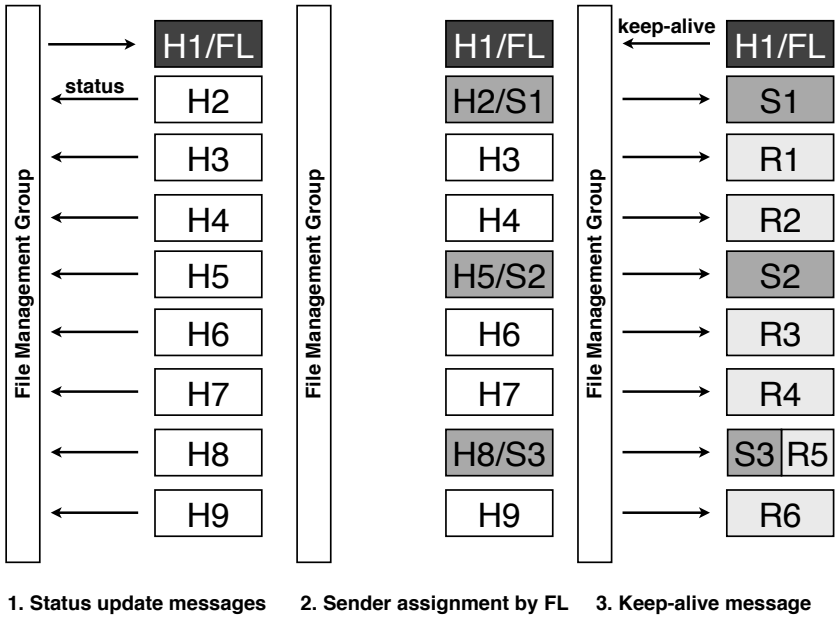


Figure 5.1: FileManagementGroup Communication in cMCFTP

For the decentralized approach dMCFTP (without a FileLeader), the FMG is used by all nodes for announcements. Nodes announce which chunks they are sending to which multicast group. All nodes listen to the announcements and then join matching sending groups for their missing chunks to be sent at supported rates. Nodes also decide which chunks they announce and send themselves. Therefore, they look at the announcements made by the other nodes. Depending on the strategy used to determine the chunks to be sent, the effectiveness can be greatly improved. Such strategies may send the least often announced chunks. Also, nodes can send chunks that have not been announced for a certain time. This helps to reach as many nodes as possible as potential downloaders and not to let some nodes starve.

5.2.4 SendingGroups

A SendingGroup (SG) is a multicast group used to transmit a chunk with a specific sending rate. The sender for a chunk sends the chunk data to this SG. All the nodes

that want to receive that chunk can join this SG. The node that has to send the chunk at the specified rate receives this information from the FileLeader. This is done via keep-alive messages transmitted via the FMG in cMCFTP mode. The FileLeader knows the available sending capacities at the determined senders. Hence, it does not assign sending rates that cannot be supported by the senders.

In the decentralized dMCFTP mode, each node announces its own intention about sending chunks and the sending rate. A node decides this by listening to what other nodes announce on the FMG. Nodes analyze which chunks other nodes send as well as which chunks are actually needed by others. Nodes that are interested in a chunk know via the keep-alive messages at which rates these chunks are sent. Therefore, nodes can join the appropriate SG that supports their free download bandwidth.

Figure 5.2 shows a cMCFTP scenario. Nodes interested in a certain chunk join the corresponding SendingGroups. The SGs have been communicated using keep-alive messages sent by the FileLeader.

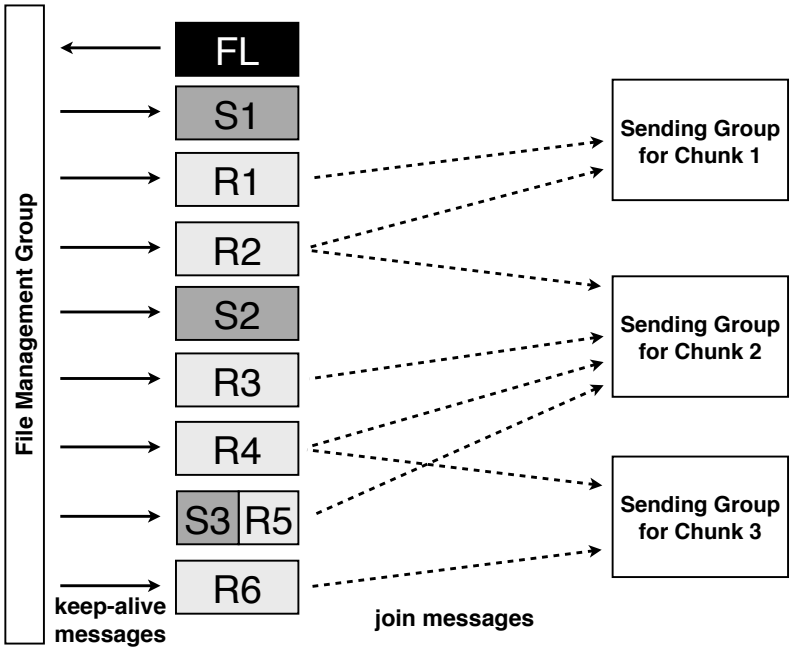


Figure 5.2: Joining SendingGroups in cMCFTP

Figure 5.3 shows how the nodes that have been announced by the FileLeader start sending data to the corresponding SGs. The nodes subscribed to the corresponding multicast groups then receive chunk data sent by the announced nodes.

IP Multicast is based on UDP and does not guarantee any reliable transmission. Therefore, erasure codes [141] or forward error correction (RFC3453) mechanisms could be applied to the SG data distribution protocol. This might add additional

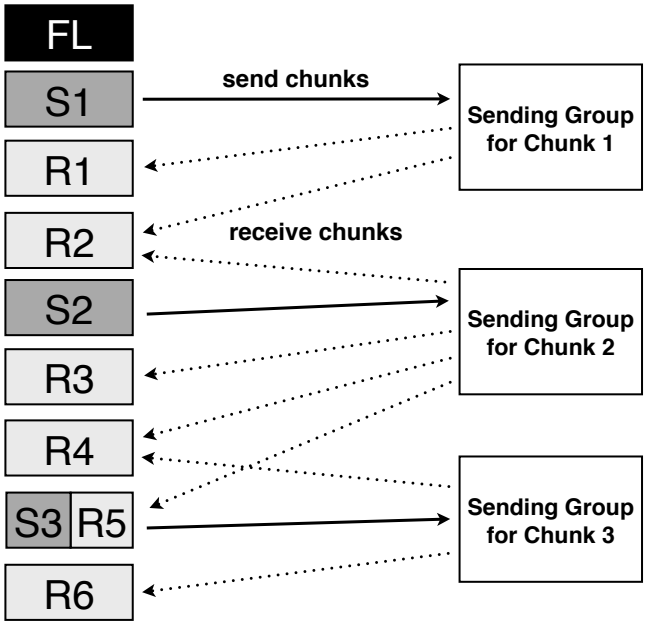


Figure 5.3: Sending and Receiving using SendingGroups in cMCFTP

data that has to be transmitted (typically around 5%). But, retransmission of certain chunks can be completely avoided if one node was not able to reconstruct the full chunk because of some packets missing.

5.2.5 Strategies to Determine SendingGroups

SendingGroup Creation Algorithm for cMCFTP in NS2 Implementation

For cMCFTP, the FileLeader determines SendingGroups for all nodes in the swarm. In order to have a well performing and efficient MCFTP swarm, all nodes should be kept busy (as senders or receivers) as often as possible. SendingGroups should be distributed evenly among all nodes participating in the swarm.

Chunks that are only available at one node should have the highest priority. It is important that such high priority chunks are distributed as quickly a possible to other nodes. Hence, the FileLeader creates for such high priority chunks corresponding SendingGroups with sending rates of 80% of the maximum available download rate in the swarm. If the sender for a rare chunk is not able to support the 80% of the maximum available download rate, the FileLeader assigns the fully reported upload sending rate to the provider of the rare chunk using a corresponding SendingGroup. This special treatment of rare chunks quickly spreads the availability of these chunks and does ensure that these chunks can also be downloaded using different rates that are potentially offered by their new providers.

If there are no more high priority chunks, the SendingGroup creation algorithm of the FileLeader targets in a first step nodes that have not yet any SendingGroups assigned. In a second step, nodes that have more than 20% of their maximum upload bandwidth unused are the target of the SendingGroup creation algorithm. Chunks are then assigned to SendingGroups depending on their availability in the targeted node, the rarity in the swarm as well as how long it has been since the chunk has been previously sent by any peer.

SendingGroup Creation Algorithm for dMCFTP in NS2 Implementation

For dMCFTP, each node individually determines its own SendingGroups. Here, the SendingGroup creation algorithm in each node is is based on history information for each chunk collected by the node itself. Nodes track how often a chunk was sent and the time a chunk has been last announced. Also, a node tracks how often it has sent a chunk and at what time this has occurred last.

There are three different priority categories for chunks. The highest priority have chunks, which have never been sent. The next lower priority category contains chunks, which have been sent by a node at least once, but have not been tracked as being offered by other nodes. The lowest priority includes chunks that have been announced by other nodes.

Chunks of the highest priority have to be distributed as quickly as possible. Hence, nodes try to send these chunks using a high sending rate, which is set to 75% of the so far maximum tracked download rate in swarm. If a node is not able to send the chunk using the determined rate, it will send the chunk using its maximum possible sending rate. This special treatment of high priority chunks again helps to quickly spread the availability of rare chunks and enable other providers to offer these chunks with variable sending rates. When a node joins the swarm, all chunks the node already has are in the highest priority category. These priorities for supposedly rare chunks though quickly decrease, as more history information is collected.

After all chunks in the highest category have been sent at least once, chunks from the second priority category are targeted by the SendingGroup creation algorithm. A node first selects the chunk, which has been sent last the longest time ago. These chunks are sent with sending rates around half of the maximum tracked download rate in the swarm. Again, if the node cannot send it at the determined rate, it will send it using it maximum possible sending rate.

Finally, chunks from the last category are targeted. These chunks are also available at other nodes in the swarm. After some time has passed, chunks are mainly placed in this last category. Hence, the focus is on continuously providing these chunks but not at very high sending rates. Chunks are chosen depending on the time they were last sent in the swarm, prioritizing the ones that have not been sent for a long time. The sending rates are set between half of the minimum observed download bandwidth in the swarm and the upload capacity of the peer. These chunks are sent twice in a row.

SendingGroup Creation Algorithm for cMCFTP in Prototype

In cMCFTP, the FileLeader receives status update messages from all nodes in the swarm and then tries to apply a strategy in order to provide efficient data dissemination in the swarm. The FileLeader stores each missing chunk information received by a node in a table, which also includes the maximum available bandwidth of the node requiring the chunk.

The basic strategy of the FileLeader works as follows. Each chunk, which is required by at least one node and can be provided by at least one other node will be announced by the FileLeader. This selection of chunks is done randomly. The FileLeader also stores how often a chunk is required and prioritizes chunks that are required more often.

In order to enhance performance, the following more advanced strategy extensions could be considered. An advanced strategy would be aware of bandwidth capability and availability of the different nodes in the swarm. Nodes that have not been selected often to disseminate chunks will also be prioritized to be used as potential senders in order to distribute the load more evenly among the swarm members. The advanced strategy tries to prefer nodes with high available sending bandwidth to send chunks that are required most. Nodes with lower available bandwidth are used to disseminate chunks, which are not required that often. We assume that less often required chunks are available at many nodes, resulting in more variable bandwidth options for sending these chunks. Also, chunks required by a small number of swarm members that can then finish their complete download of the file in order to become seeders are also prioritized. Furthermore, chunks that are required most in the MCFTP swarm could also be prioritized.

We implemented the basic strategy and included some optimizations proposed for the advanced strategies (such as bandwidth awareness and prioritization).

SendingGroup Creation Algorithm for dMCFTP in Prototype

In dMCFTP, nodes decide by themselves which chunks they want to send. Each node also follows a strategy to select the chunks to be sent.

Using a basic strategy, nodes collect status messages and find out which chunks are required by other nodes in the swarm. Then, a node can determine which of the chunks it already has can be potentially offered to other nodes. It chooses a random chunk out of this list of potential chunks to be sent, and uses its fully available upload bandwidth to send the chunk. As a variation, a node could randomly select two chunks and send each with half of its available upload bandwidth.

More advanced strategies would also include bandwidth awareness of other nodes acting as potential receivers as well as potential senders. This would facilitate that in the swarm, the same chunk would be sent with various sending rates. Nodes could all follow the same strategy or different strategies depending on their download completion state (seeder, no chunk, 50% of chunks, etc.) or bandwidth capabilities. A mixture of strategies could be beneficial for overall performance.

We implemented the basic strategy and included some simple optimizations (such as bandwidth awareness and comprehension of sending activities of other nodes) and prioritizing mechanisms for our prototype.

5.3 Evaluation Scenarios for MCFTP

5.3.1 Overview

We evaluated MCFTP using the network simulator ns2 and also using a prototype implementation. The prototype implementation was evaluated in a local testbed.

MCFTP was compared with BitTorrent in both environments. In the network simulator, we implemented the basic BitTorrent protocol. We then compared our MCFTP implementation with this BitTorrent implementation in ns2 regarding performance. For the comparison with the prototype implementation of MCFTP, we used the popular Azureus [12] BitTorrent client.

The evaluation in the network simulator used small to large scale networks up to 2041 nodes, whereas the prototype was only evaluated in small scale networks with up to 100 nodes. Section 5.3.2 will describe the scenarios used to evaluate MCFTP using the ns2 network simulator. The evaluation using the prototype implementation is presented in Section 5.3.3.

5.3.2 NS2 Network Simulator Evaluation Scenarios

Overview

To make a first evaluation of MCFTP, we implemented the protocol using the ns2 [70, 29, 65, 92, 119] simulator. We compared the performance of the different MCFTP approaches (cMCFTP and dMCFTP) with BitTorrent presented in Section 2.4.7. The evaluations were performed using small to large scale networks, ranging from 33 to 2041 nodes for IP Multicast and from 37 to 525 nodes for Overlay Multicast. We present the different simulation scenarios for IP Multicast and Overlay Multicast. This includes details about network topologies, network sizes and the join behavior of nodes for MCFTP and BitTorrent in the simulation runs.

Simulation Scenarios

All simulation scenarios except for the impact of seeders on download duration factor look at the worst case scenario, where initially only one seeder (a node having the full file) is available. Nodes that want to download the file arrive over time in the network and join the swarm. This is often also referred to as flash crowd scenario. In all simulation scenarios, seeders (initial seeders and nodes that develop to seeders over time) remain active in the swarm until the simulation has ended.

Simulations were performed using IP Multicast as well as using Overlay Multicast environments. In the Overlay Multicast environment, we used a shared overlay

network for all multicast groups with two scenarios. For the first scenario, the overlay network was constructed to support QoS. Therefore, a parent node can fully support the maximum possible required bandwidth of the children nodes attached. To support QoS, parents reserve some of their bandwidth for all their children. Therefore, we used a bandwidth scale factor of 0.5 for the first scenario. This means that a parent reserves half of its own download bandwidth for its children to support QoS. The other half is used by the parent itself. This way a node can guarantee that it always has enough bandwidth to serve its own download requests as well as all download requests coming from nodes in its attached sub-tree. If the parent and all its children on the whole sub-tree use their full bandwidth, all nodes including the parent can be served. The parent's bandwidth is never exceeded. Hence, there should be no packet loss in the first scenario.

In the second scenario, we used an overlay network supporting only partially the bandwidth requirements of attached children. Therefore, we chose a bandwidth scale factor of 0.75 for the second scenario. Here, a parent assigns up to 75% of its possible bandwidth to itself. But still, 50% of the parent's bandwidth is assigned to the direct children nodes. In case the parent and all its direct children would use their fully assigned bandwidth, the parent's bandwidth available on the link could not support the totally required bandwidth. The link would actually require 125% of its available bandwidth. This effect increases with each additional level in the sub-tree. Therefore, a node cannot guarantee that all children and itself are able to use their full bandwidth capabilities.

Network Topologies, Network Sizes, File Sizes and Join Behavior

The network topologies for the IP Multicast simulation scenarios were constructed as follows. The random number generator influences different values determining the topology setup. The core router network is though manually predefined and is the same for each topology. Attached to each of the routers in the core router network, a tree consisting of routers is randomly built. The depth of each tree is randomly determined using a normal distribution. The number of routers in the tree is determined randomly as well. Delays of the links between routers in the trees are also determined randomly using a normal distribution. Bandwidth capabilities of the links are set high in order to avoid congestion. Nodes running BitTorrent or MCFTP applications are then connected to the various routers in the sub-trees. The delay and bandwidth capabilities of the links connecting the nodes to routers are also determined randomly. The bandwidth capabilities assigned to the links is asymmetric (different upload and download bandwidth) and determined by a ratio parameter. We set the ratio parameter between upload and download bandwidth to 0.3 (upload = $0.3 \times$ download) and the mean of the randomly determined bandwidth capabilities distribution on the links was set to 2000 Kbps with a standard deviation of 550 Kbps and a minimum of 384 Kbps. Also, a rendez-vous point is required to support IP Multicast in the simulated topology. The rendez-vous point is always at a fixed position in the core router network.

For the overlay network scenarios, the topologies were built using the following approach. Compared to the IP Multicast topologies, we use symmetric links for the nodes, and the end-to-end delays are also higher. The generated topology acts as a shared tree for the whole overlay network. We had to build the Overlay topologies differently, due to the fact that we needed to guarantee certain QoS requirements in terms of bandwidth and to avoid congestion. There are two different node types, simple data forwarders, which consist only of a simple node in the simulator, and applications consisting of two nodes in the simulator. For the applications, one node in the simulator is used as multicast data forwarder while a second node contains the application. Those two nodes in the simulator are connected by a link with a delay of 0ms. If the node uses BitTorrent, the link bandwidth is limited to support TCP-connection scaling. In case MCFTP is used, the link bandwidth is very high because MCFTP regulates the sending rate by itself.

The construction of the topology resulting in the shared overlay tree is then finally constructed as follows. The root of the tree has a specified start bandwidth. The number of children for each node (including the root) is determined randomly. In case the current node is only acting as forwarder, the available bandwidth (coming from the parent link) is distributed randomly among all links connecting the children nodes. If the node acts as application, then only 50% of the parent link bandwidth capacity is assigned randomly to the links connecting the children nodes. In case of the QoS scenarios with a bandwidth scale factor of 0.5 (where applications would only assign 50% of the available bandwidth capacity to themselves), QoS in terms of required bandwidth would be fully supported. But, if applications would assign 75% of their bandwidth capacity to themselves (using a bandwidth scale factor of 0.75), then the network would be over-used and congestion would occur. Also, the rendez-vous point would need to be placed in the root of the constructed overlay tree topology.

We compare different network sizes with node numbers ranging from 33 to 2041. This covers small groups of nodes exchanging a file of common interest as well as large groups for exchanging more popular files. The topologies for the different network sizes were built independently as described, before leading to different characteristics depending on the number of nodes.

The simulations use different file sizes (50 MB and 100MB). Each scenario is simulated with different independent runs. For each of the simulations, the arrival times of nodes, the placement of the seeder or seeders, the placement of the Tracker (for BitTorrent) or FileLeader (for MCFTP), and the initialization of the random number generator differs. Using different placements for the seeder(s) leads to different transfer rates (upload and download) for these nodes. Also, the Tracker and FileLeader are placed at different nodes for each simulation. For dMCFTP, the particular node that normally acts as Tracker or FileLeader remains empty (not participating in the swarm), in order to keep the results comparable.

Nodes join the swarm uniformly and randomly distributed over time using a specified start period. For the IP Multicast scenarios, this period is between 2–1500s for a file size of 50 MB and between 2–2000s for a file size of 100 MB. For

the Overlay Multicast scenarios, this period always starts at 2s and ends, depending on the number of nodes, between 5000–7000s for a file size of 50 MB and between 10000–12000s for a file size of 100 MB. The same arrival times are used in the different scenarios comparing MCFTP with BitTorrent. There is normally only one node in the network that has the full file (called seeder). This is usually the first joining node. The simulations end after a simulated runtime of 3500s for a file size of 50 MB and after 5500s for a file size of 100 MB.

5.3.3 Prototype Implementation Evaluation Scenarios

Overview

To compare the performance of MCFTP with an existing and popular BitTorrent client, we implemented a MCFTP prototype using JAVA. Using a local testbed at University of Bern, we then compared MCFTP and the BitTorrent client Azureus [12] regarding performance.

The evaluation scenarios consists of small network sizes ranging from 10 to 100 nodes. We performed the evaluations using the RVS research group's cluster, which consists of 25 nodes with different hardware configurations. The following different configurations are available in the testbed:

- Pentium D 3.0 GHz or 3.2 GHz PCs with 1 GB, 2 GB or 3 GB RAM
- AMD Athlon Opteron 250 2.4 GHz Dual-CPU PCs with 4 GB RAM
- AMD Athlon 64 X2 Dual Core 4800+ 2.5 GHz PCs with 6 GB RAM
- AMD Athlon 64 X2 Dual Core 5200+ 2.6 GHz PCs with 2 or 6 GB RAM
- Core2Duo 3.33 GHz PCs with 16 GB RAM
- Core2Quad 2.86 GHz PCs with 8 GB RAM

All nodes run a remotely booted Fedora Core 8 Linux for 64 Bit. They do not have anything locally installed, the integrated hard drives (80–500GB) are only used as swap space and for temporary files. The nodes were interconnected using three Gigabit Ethernet Switches. Each node hosted one or more instances of our MCFTP prototype or Azureus BitTorrent client, depending on the nodes in the scenario.

Simulation Scenarios

All simulation scenarios except the one comparing the impact of seeders on download duration factor used a flash crowd scenario with initially having only one seeder. Nodes that want to download the the file arrive over time in the network. This is often also referred to as flash crowd scenario In all simulation scenarios, seeders (initial seeders and nodes that develop to seeders over time) remain active in the swarm until the simulation has ended.

The evaluations in the local testbed were performed using IP Multicast as well as using Overlay Multicast. To support Overlay Multicast, Freepastry was used to integrate Scribe/Pastry ALM functionality into the MCFTP prototype. We used one Scribe/Pastry instance for the FMG and another shared instance for all the SendingGroups. Therefore, we were able to separate communication message exchange and file dissemination traffic. Each scenario was run 10 times and we removed 5% of the outliers.

We compare dissemination of files with a size of 8 MB and 50 MB. A simulation in a scenario for files of 8 MB size usually was running for 30 to 45 minutes depending on the number of nodes in the swarm. The simulation runs in scenarios with 50 MB files took up to 1.5h depending on the number of nodes. This runtime includes deployment, startup, and result collection.

Network Characteristics, Network Sizes, File Sizes and Join Behavior

We configured the applications (MCFTP and BitTorrent) to use asymmetric links. The download bandwidth for nodes ranged between 60 KBps and 180 KBps and the upload bandwidth corresponded to $\frac{1}{3}$ of the download bandwidth assigned. The bandwidth assignments were distributed uniformly among participating nodes.

The number of nodes in the different simulation scenarios was normally set to 10, 20, 30, 50 and 100 nodes. The simulation scenarios with a file size of 50 MB were not evaluated with 100 nodes due to the long runtime of the individual simulations. For the simulation scenario comparing the impact of seeders on download duration factor we varied the number of initial seeders between 1, 2, 3 and 5 with a total of 20 and 50 nodes in the swarm.

We distributed one file at a time and evaluated the scenarios with two different file sizes. The first file was a Firefox tar-ball with a file size around 8 MB, the second file was a disk image of the "Damn Small Linux" distribution with a file size of approximately 50 megabytes. Nodes also join the network randomly over a certain time period. We used the same arrival times for MCFTP and BitTorrent scenarios. Normally, only one node in the network has the full file (called seeder), which is usually the first joining node. Nodes remain in the swarm after they joined.

5.4 Evaluation Results for MCFTP

5.4.1 Overview

MCFTP was evaluated in two different environments. We used the network simulator ns2 to compare MCFTP to BitTorrent in small to large scale networks up to 2041 nodes. The prototype implementation was compared to Azureus, a popular BitTorrent client, in small scale networks with up to 100 nodes.

We present the results of the evaluation of MCFTP using the ns2 network simulator in Section 5.4.2. The results of the evaluation using the prototype implementation are presented in Section 5.4.3.

5.4.2 NS2 Network Simulator Evaluation Results

Overview

We evaluated various aspects of MCFTP and compared those with BitTorrent where applicable. We first compare the download duration factor for MCFTP using IP Multicast with BitTorrent. Then, we present the impact of seeders on the download duration factor. We also compare upload and download bandwidth consumption of the FileLeader with cMCFTP using IP Multicast. The comparison of the development of seeders over time is also presented. Thereafter, we compare the bytes transferred over time. Then, we present the comparison of the download duration factor for MCFTP and BitTorrent in Overlay Multicast environments. For all above mentioned comparisons, we used the same set of simulations / experiments, except for comparison of the impact of seeders on the download duration factor which is a separate dedicated simulation / experiment where we varied the number of seeders available at the beginning of the simulation. Finally, we compare MCFTP with IP Multicast and Overlay Multicast. For each network size, we usually built 20 different topologies. In the evaluation results presented, we removed 5% of the outliers. Details of the simulation scenarios setup, as well as what files (in terms of size) are distributed how in the simulations were described in Section 5.3.2.

Download Duration Factor

The download duration factor measures the ratio between the maximum bandwidth available at a node and the bandwidth that was effectively achieved during the download of a file. The download duration factor $df \geq 1$ is defined as $df = \frac{dl_{eff}}{dl_{max}}$, with $dl_{eff} > 0$ representing the effectively achieved download bandwidth, $dl_{max} > 0$ representing the maximum available bandwidth (given by the Internet connection of a node), and $dl_{eff} < dl_{max}$.

As an example, if the effective bandwidth achieved during a download is, e.g., 1 Mbps and the node is connected to the Internet using a 2 Mbps connection, then the resulting download duration factor is 2. This symbolizes that the download was actually twice as long as it would have been for the best case, when the full available bandwidth of the node would have been used. Hence, a larger download duration factor results in a lower utilization of the available bandwidth. Therefore, the best achievable download duration factor is 1, which corresponds to the maximum exploitation of available bandwidth of a node.

Figures 5.4 and 5.5 compare the download duration factor (average, minimum and maximum) for BitTorrent and MCFTP using IP Multicast.

In Fig. 5.4, the download duration factor for a file of 50 MB is presented. MCFTP performs always better than BitTorrent in networks with 33 to 2041 nodes. The difference between dMCFTP and cMCFTP is not significant.

For a file of 100 MB, the results are similar as shown in Fig. 5.5. MCFTP performs again always better than BitTorrent. But the differences are smaller for large network sizes as compared to downloading a file of 50 MB.

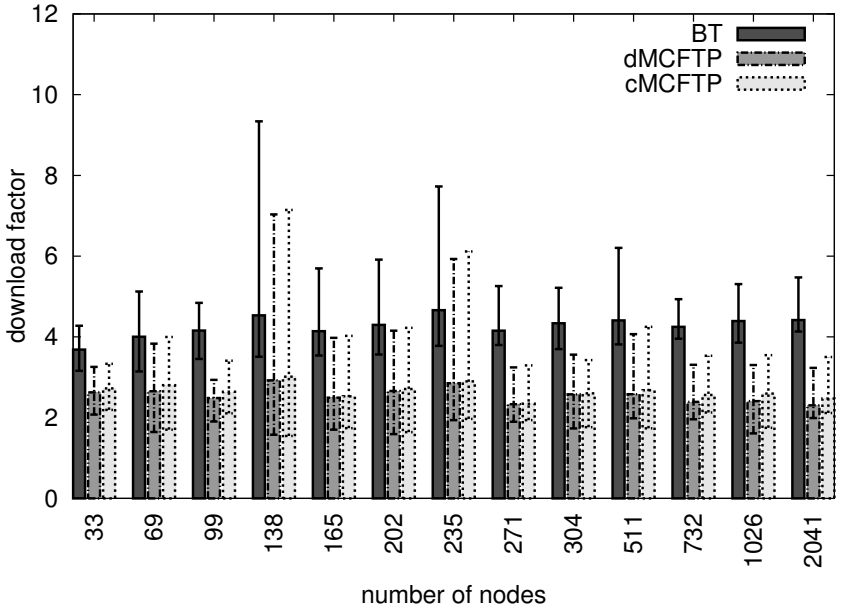


Figure 5.4: MCFTP and BitTorrent Download Duration Factor for a 50 MB File

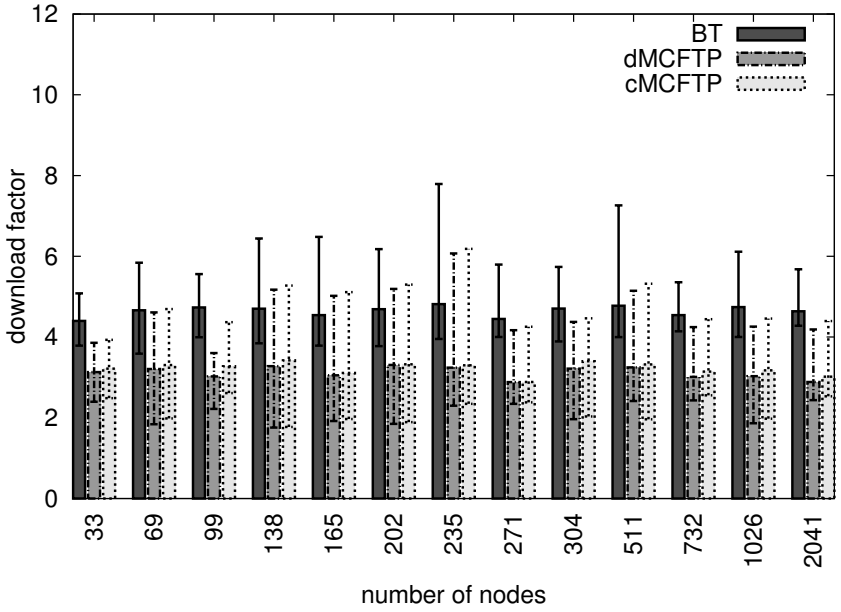


Figure 5.5: MCFTP and BitTorrent Download Duration Factor for a 100 MB File

Impact of Seeders on Download Duration Factor

A seeder is a node that already has the complete file. Therefore, it can provide all chunks of the file to potential downloaders. We now compare the impact of the

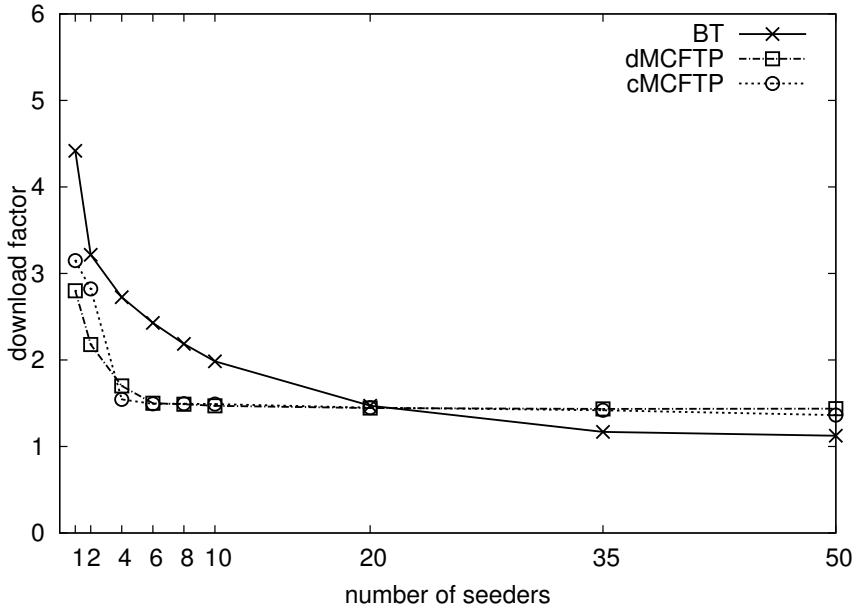


Figure 5.6: Impact of Seeders on Download Duration Factor for MCFTP and BitTorrent in 69 Node Scenarios

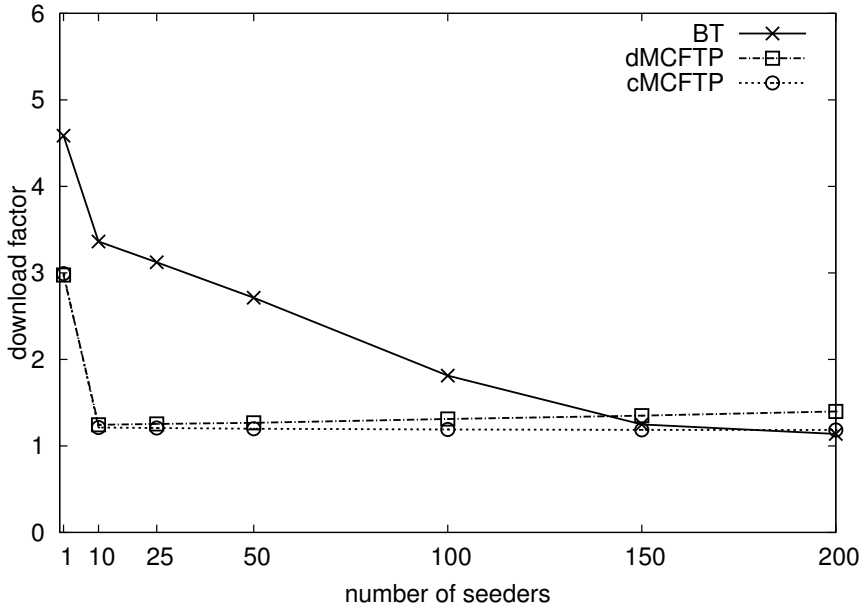


Figure 5.7: Impact of Seeders on Download Duration Factor for MCFTP and BitTorrent in 304 Node Scenarios

number of seeders (available from the beginning of the simulation) on the download duration factor for dMCFTP and cMCFTP using IP Multicast. We calculated

the average over the download duration factor for file sizes of 50 MB and 100 MB.

Figure 5.6 shows the impact of the number of seeders in a scenario with 69 nodes. MCFTP performs better than BitTorrent for scenarios with up to 20 seeders. The download duration factor is lower using MCFTP than using BitTorrent. Therefore, nodes can download the files faster using MCFTP than using BitTorrent when up to 20 seeders are available from the beginning of the simulation run. But, there is no big difference between dMCFTP and cMCFTP.

Figure 5.7 shows the impact of the number of seeders in a scenario with 304 nodes. MCFTP performs better for scenarios with up to 140 seeders.

Generally, MCFTP performs better for low numbers of seeders. It reaches its optimum already quite early when only having very few seeders. BitTorrent on the other hand performs better with a high number of seeders, which is though not the normal case. It also converges much slower to its optimum. MCFTP's strategy tries to first create as many additional seeders as possible, which has a positive impact on the overall performance in scenarios with a low number of seeders available at the beginning of a simulation scenario. Only afterwards it resumes normal file distribution operation whiteout heavily prioritizing seeder creation. With more seeders available, BitTorrent has the advantage that it can more individually serve nodes that require missing chunks. Hence, when at least 30–40% of the nodes are seeders from the beginning of a simulation scenario, then BitTorrent performs better than MCFTP.

Upload and Download Rate in cMCFTP

Now, we compare upload and download bandwidth consumption of the FileLeader with cMCFTP using IP Multicast. We compare the mean and maximum values. This shows whether the FileLeader with its limited Internet connectivity can become a possible bottleneck for the whole system. It also shows how well cMCFTP scales.

The download bandwidth consumption at the FileLeader is determined by the accumulated sum of the status update messages. The upload bandwidth consumption at the FileLeader is caused by keep-alive messages sent by the FileLeader. The size and sending time interval of these messages varies and depends on the number of nodes in the swarm, the progress of overall download in the swarm, the individual download speed of chunks by nodes, and therefore also the rate at which nodes have to report updates. We chose to give a dedicated role to the FileLeader in the presented simulation scenarios. This means that a FileLeader only receives status update messages and sends keep-alive messages. In the simulation scenarios, the FileLeader is not involved in the actual multicast data distribution. This could though be handled differently in real scenarios. Figure 5.8 shows the upload and download bandwidth consumption (in bytes per second) in a scenario where a 50 MB file is disseminated. The upload bandwidth consumption grows only slowly for scenarios with more nodes and is becoming constant after a certain number of nodes. This is due to the fact that we limit the rate of keep-alive messages. There-

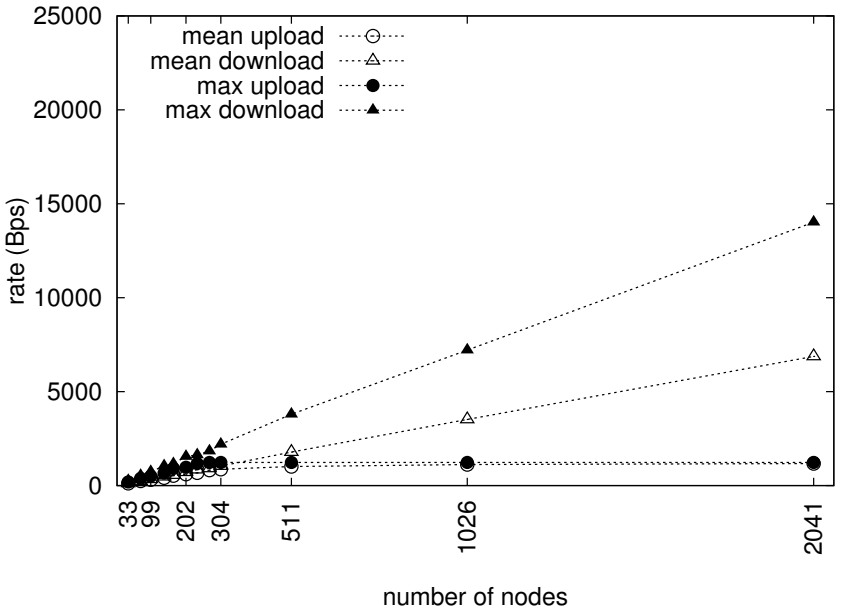


Figure 5.8: cMCFTP Upload and Download Rates of the FileLeader for a 50 MB File

fore, MCFTP messages consume only a small fraction of the incoming bandwidth of participating nodes. The download of the FileLeader on the other hand increases with the number of nodes in the scenarios. The more nodes are in the system, the more status update messages consume incoming bandwidth on the FileLeader. The rate of 120 Kbps (15000 bytes per second) total incoming traffic for scenarios of 2041 nodes is though still easily manageable by the FileLeader.

For a scenario where a 100 MB file is disseminated as shown in Fig. 5.9, the values for the download are higher. This is due to the fact, that a file size of 100 MB results in a higher number of chunks for the file. This also means that status update messages are then longer. This is because information of more chunks has to be reported to the FileLeader by participating nodes. The upload remains the same due to the keep-alive message limitation as mentioned before.

Seeder Development over Time

We now compare the development of seeders over time for MCFTP using IP Multicast and BitTorrent. If a node has successfully downloaded the whole file, it becomes a seeder. Therefore, when all nodes are seeders, the whole swarm of nodes has downloaded the file.

Figure 5.10 shows the seeder development in a scenario with 165 nodes downloading a file of 100 MB. After the simulation has run for 2000 seconds, all nodes have joined the downloading swarm. Using MCFTP, downloads finish earlier. There is no difference between dMCFTP and cMCFTP, the lines for dMCFTP and cMCFTP in the graph cover each other.

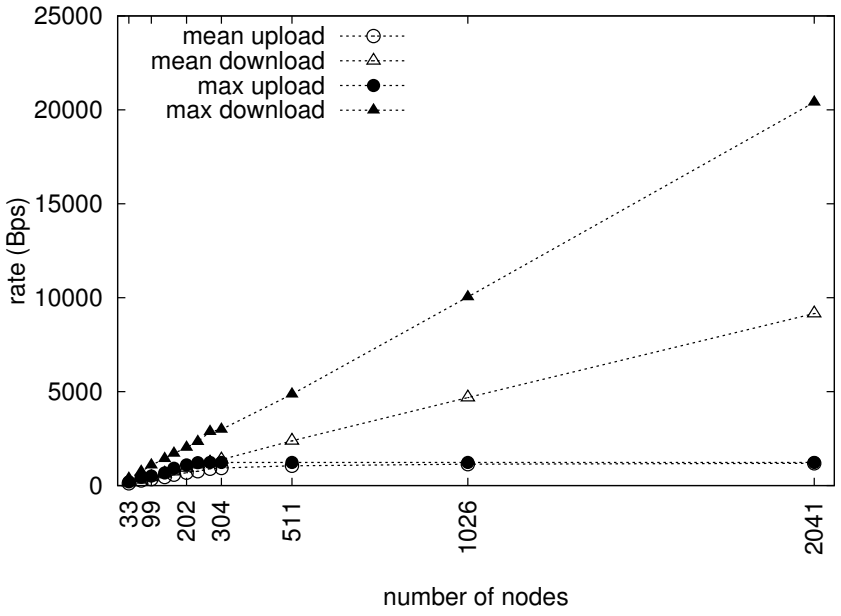


Figure 5.9: cMCFTP Upload and Download Rates of the FileLeader for a 100 MB File

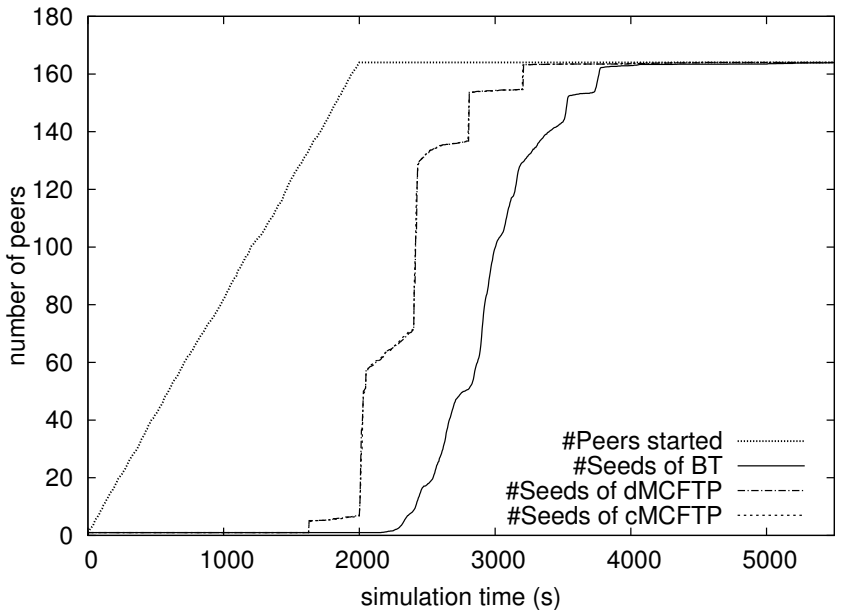


Figure 5.10: Number of Seeders over Time for MCFTP and BitTorrent with a 100 MB File and 165 Nodes

Figure 5.11 shows the seeder development in a scenario with 511 nodes downloading a file of 50 MB. In this scenario, all nodes have joined the swarm after 1500 seconds. Again, MCFTP performs better than BitTorrent. There is only a

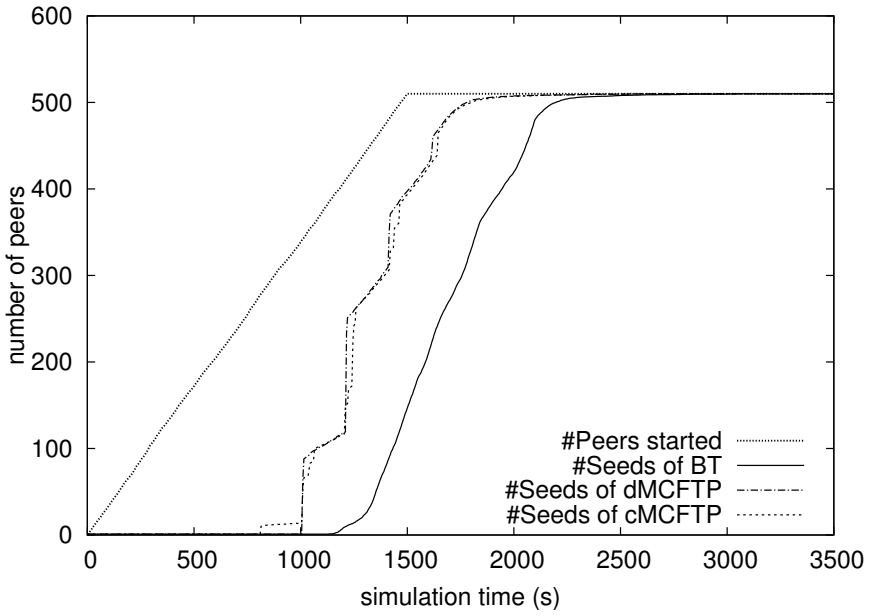


Figure 5.11: Number of Seeders over Time for MCFTP and BitTorrent with a 50 MB File and 511 Nodes

slight difference between cMCFTP and dMCFTP. MCFTP is prioritizing to create additional seeders in the swarm as quickly as possible. Hence, the faster additional seeders are available, the more options are offered to other nodes from where they can download their missing chunks. This means that new seeders develop faster and earlier when using MCFTP instead of BitTorrent. This is due to the fact that BitTorrent does not try to create new seeders as early and as aggressively as MCFTP does. It rather tries to equally download rare pieces first and independently of the download bandwidth available at the providing node, which is potentially the only seeder or even the only node having any chunks at all available in the swarm. Therefore, seeders develop generally slower over time when using BitTorrent than when using MCFTP.

Bytes Transferred over Time

We now compare the bytes transferred over time (including file transmission data and protocol data) for BitTorrent and MCFTP using IP Multicast. The total upload and total download values represent the maximum possible upload and download rates. This is the sum of all rates of the participating nodes.

In Fig. 5.12, the rates for a 100 MB file in a scenario with 165 nodes are shown. The behavior of the rates is similar to Fig. 5.13, which shows a scenario for a 50 MB file with 511 nodes.

The download that can be achieved using dMCFTP actually exceeds the overall maximum possible upload of all nodes. This means that the accumulated upload

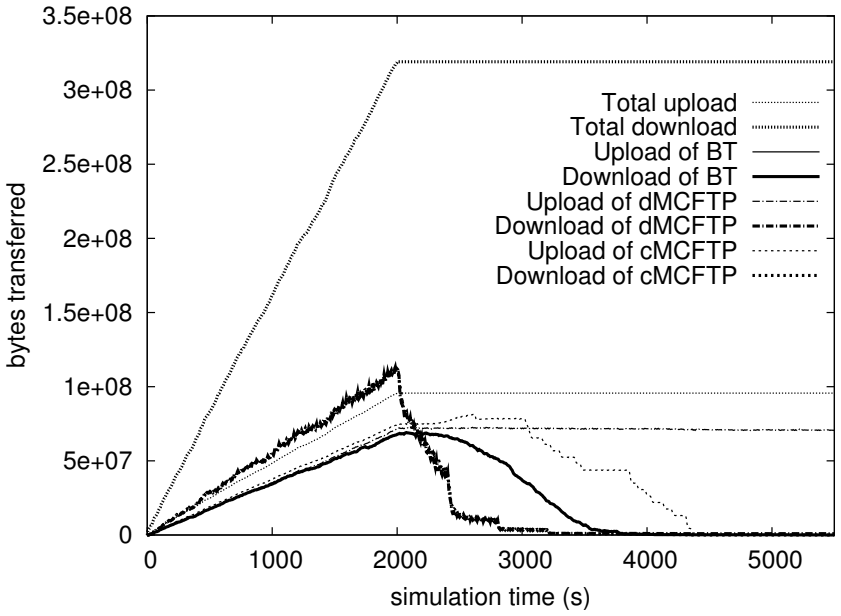


Figure 5.12: Bytes Transferred over Time for MCFTP and BitTorrent with a 100 MB File and 165 Nodes

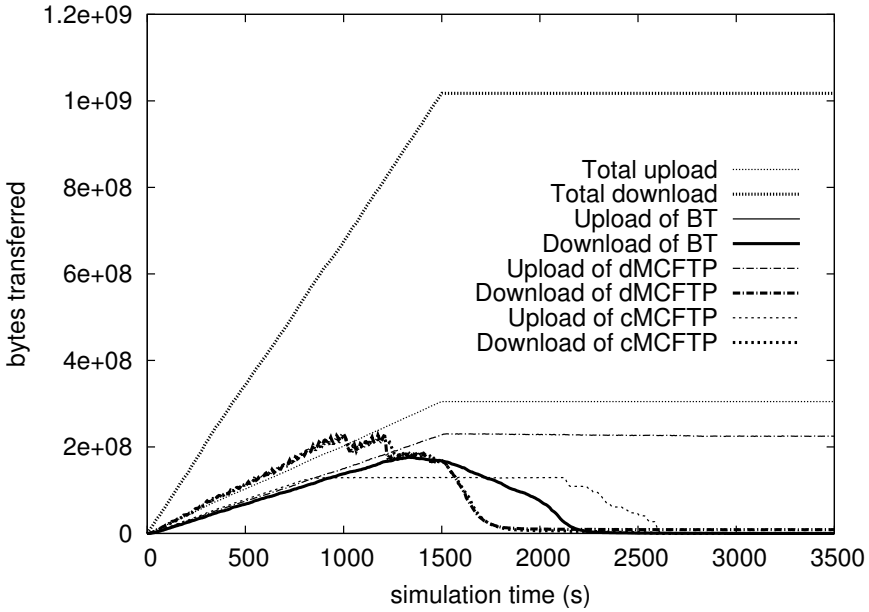


Figure 5.13: Bytes Transferred over Time for MCFTP and BitTorrent with a 50 MB File and 511 Nodes

bandwidth capabilities of all nodes is lower than the accumulated total download bandwidth effectively achieved by all nodes.

This is due to the fact that MCFTP uses IP Multicast. But for cMCFTP, the download does not exceed the maximum possible upload rate. Because cMCFTP has a FileLeader that has an overview of all nodes' requirements, downloads are distributed more evenly.

The FileLeader tries to assign SendingGroups such that many nodes with different download bandwidth capacities that have certain chunks missing can be served at the same time. Therefore, the FileLeader assigns different SendingGroups with varying sending rates for the missing chunks in parallel. This results in a more homogeneous distribution of the downloads. Therefore, on average there might be less subscribers per multicast group as compared to dMCFTP. Hence, cMCFTP can benefit less from IP Multicast distribution than dMCFTP. Upload rates on the other hand behave almost the same for both MCFTP modes.

Upload and download using BitTorrent are the same, since nodes serve each other. Every upload rate on a node results in the same download rate on another node. Therefore, the lines for upload and download for BitTorrent in the graph cover each other.

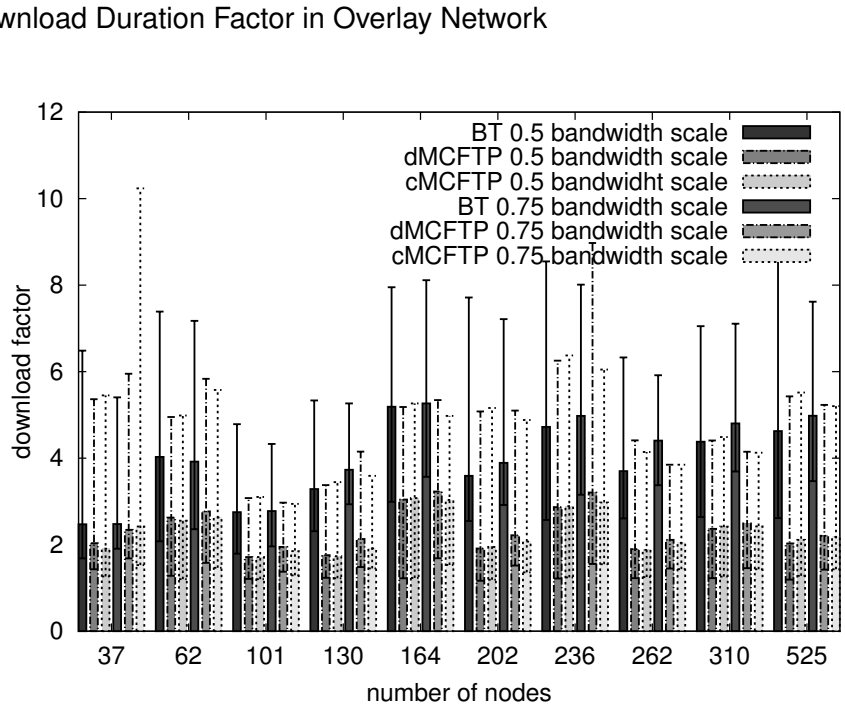


Figure 5.14: MCFTP and BitTorrent Download Duration Factor in Overlay Environment for a 50 MB File

We now compare the download duration factor (average, minimum and maximum) for BitTorrent and MCFTP using Overlay Multicast. As explained in Section 5.3.2, we looked at overlays that support QoS and possibly overloaded overlay networks. As mentioned, a bandwidth scale of 0.5 means that a parent reserves half

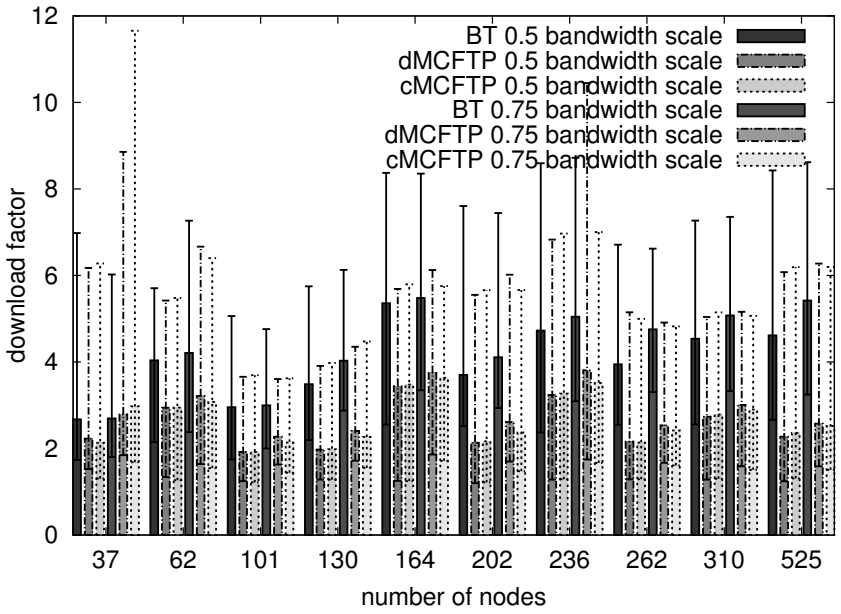


Figure 5.15: MCFTP and BitTorrent Download Duration Factor in Overlay Environment for a 100 MB File

of its own bandwidth for its children to support QoS. However, with a bandwidth scale of 0.75, a parent uses up to three quarters of its possible bandwidth for itself. Therefore, the parent's bandwidth can be exceeded by 50% and QoS can not be supported.

In Fig. 5.14, the download duration factor for a file of 50 MB is presented. MCFTP performs always better than BitTorrent in networks with sizes from 37 to 525 nodes. With more nodes, MCFTP performs significantly better than BitTorrent. The difference between dMCFTP and cMCFTP is marginal.

For a 100 MB file, the results are similar as shown in Fig. 5.15.

MCFTP with IP Multicast & Overlay Multicast

We can compare the download duration factor results for IP Multicast and Overlay Multicast by looking at Figures 5.4, 5.5 and 5.14, 5.15. The simulations for IP Multicast and Overlay Multicast were performed using different network topologies, since they have been constructed differently as described in Section 5.3.2.

Comparing the MCFTP results, we can see that MCFTP generally performs slightly worse using Overlay Multicast than IP Multicast. This is due to replication of multicast data at end systems rather than using routers in the core network. Also the shared bandwidth for forwarding traffic among peers for the Overlay Multicast scenario with a bandwidth scale factor of 0.75 increases the download duration factor for MCFTP.

Using the network topology for Overlay Multicast scenarios, for small networks (less than 164 nodes) BitTorrent is faster than using the IP Multicast scenarios topology. But for networks with 164 and more nodes, the download duration factors for BitTorrent are again quite similar between the two different topologies.

5.4.3 Prototype Implementation Evaluation Results

Overview

We evaluated different aspects comparing MCFTP with BitTorrent where applicable. First, we compare the download and upload factor for MCFTP and BitTorrent. We look at the overall download duration factor and also the download and upload factor over time. Then, we present the impact of seeders on the download duration factor. Finally, the seeder development over time is presented.

The different scenarios were run with file sizes of 8 MB and 50 MB. We look at all network sizes evaluated when analyzing the overall download duration factor. Evaluation of the remaining aspects is done by comparing the scenarios with either 20 or 50 nodes. In the evaluation results presented, we removed 5% of the outliers. Details of the simulation scenarios setup, as well as what files (also in terms of file size) are distributed how in the simulations were described in Section 5.3.3.

Download Duration Factor

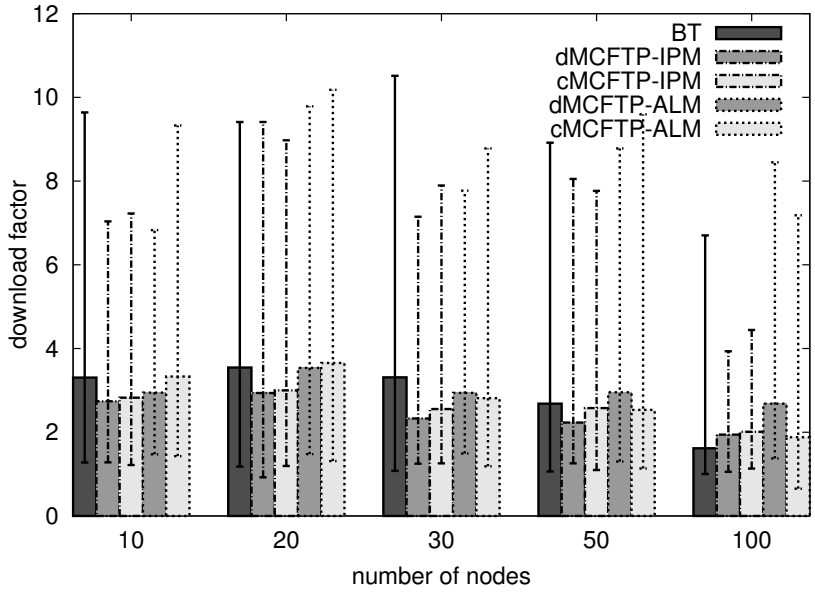


Figure 5.16: Download Duration Factor for BitTorrent, dMCFTP, cMCFTP for a 8 MB File

In Fig. 5.16, we compare the download duration factor (average, minimum and

maximum) for BitTorrent with dMCFTP and cMCFTP both using IP Multicast as well as ALM for a file size of 8 MB.

In Fig. 5.17, we compare the download duration factor for BitTorrent and dMCFTP using IP Multicast and ALM for a file size of 50 MB. We can see that

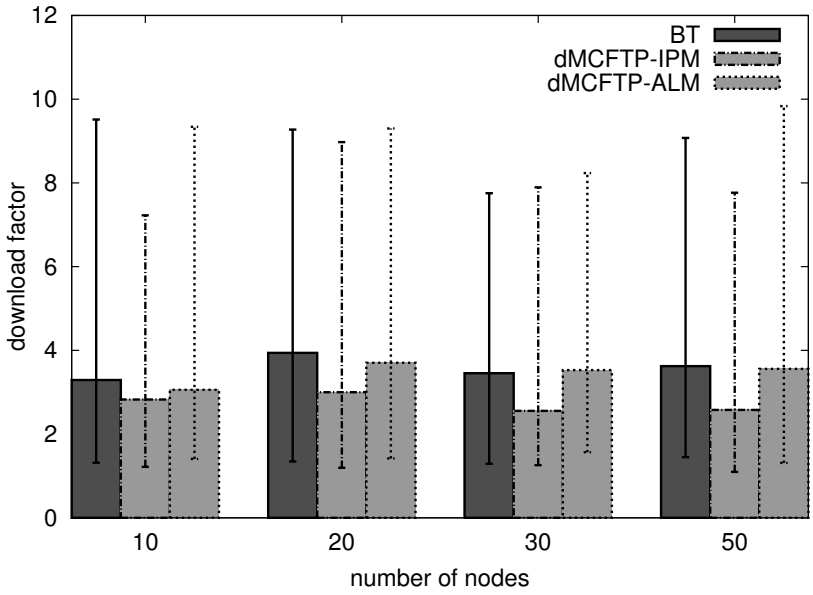


Figure 5.17: Download Duration Factor for BitTorrent, dMCFTP, cMCFTP for a 50 MB File

MCFTP using IP Multicast and ALM performs better than BitTorrent for small network sizes up to 30 nodes. For larger number of nodes with a file size of 8 MB, dMCFTP using ALM performs worse. This is due to the fact of the shared overlay network for data dissemination. Freepastry uses the object serialization and de-serialization mechanism provided by JAVA to process each multicast message/packet individually which limits the achievable overall bandwidth at a node. Also having multiple topics (groups) active in a Scribe/Pastry instance puts some limits to the scalability of the overall system.

Impact of Seeders on Download Duration Factor

We compare the impact of the number of seeders (available from the beginning of the evaluation run) on the download duration factor for BitTorrent, dMCFTP using IP Multicast, and dMCFTP using ALM. We used a file size of 8 MB for this evaluation.

Figure 5.18 shows the impact of the number of seeders in a scenario with 20 nodes. dMCFTP performs better than BitTorrent for scenarios with up to 2 seeders. The download duration factor is lower using dMCFTP with IP Multicast or ALM than when using BitTorrent. Hence, nodes download files faster using dMCFTP

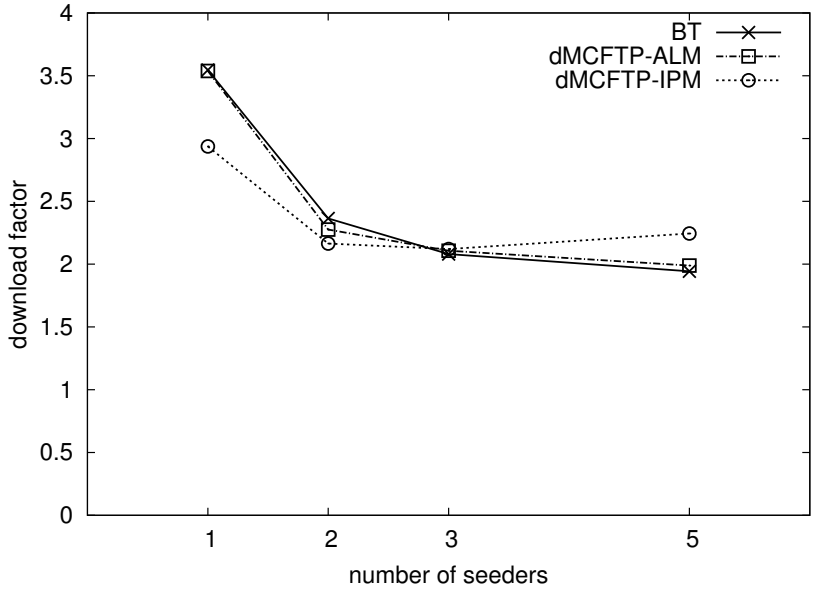


Figure 5.18: Impact of Seeders on Download Duration Factor for 20 Nodes

than using BitTorrent when there are up to 2 seeders available from the beginning of the evaluation runs. With 3 seeders, dMCFTP performs similar to BitTorrent.

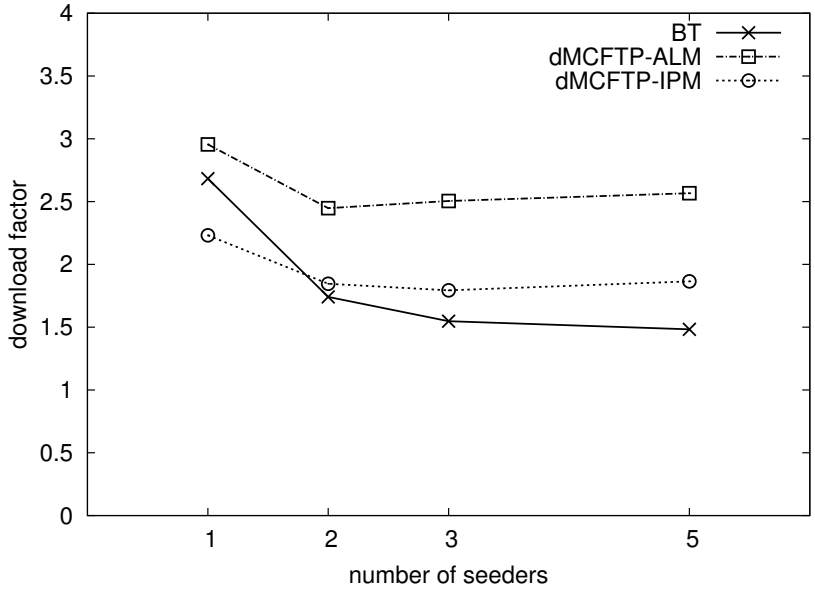


Figure 5.19: Impact of Seeders on Download Duration Factor for 50 Nodes

Figure 5.19 shows the impact of the number of seeders in a scenario with 50 nodes. Here, dMCFTP using IP Multicast performs better for 1 seeder and similar

for 2 seeders when comparing it with BitTorrent. dMCFTP using ALM already performs worse for 1 seeder. This is again due to the fact that Freepastry does not scale well for large number of nodes and concurrently active topics in one Scribe/Pastry instance. For 3 and more seeders, BitTorrent performs always better than dMCFTP.

Upload and Download Bandwidth Usage over Time

The upload and download bandwidth usage over time presents how many percent of their available upload and download bandwidth nodes on average actually use. This means that at a given point in time, a node is for example downloading some chunks with a total download rate corresponding to only half of its totally available bandwidth, therefore resulting in a download bandwidth usage of 50% at that particular moment. This is an indicator for the efficiency of the different protocols, especially when compared in relation to the overall download duration factors. If two protocols have similar download duration factors, but one of those protocols uses on average less upload and download bandwidth over time, then this protocol would be more efficient in terms of bandwidth usage and therefore perform better.

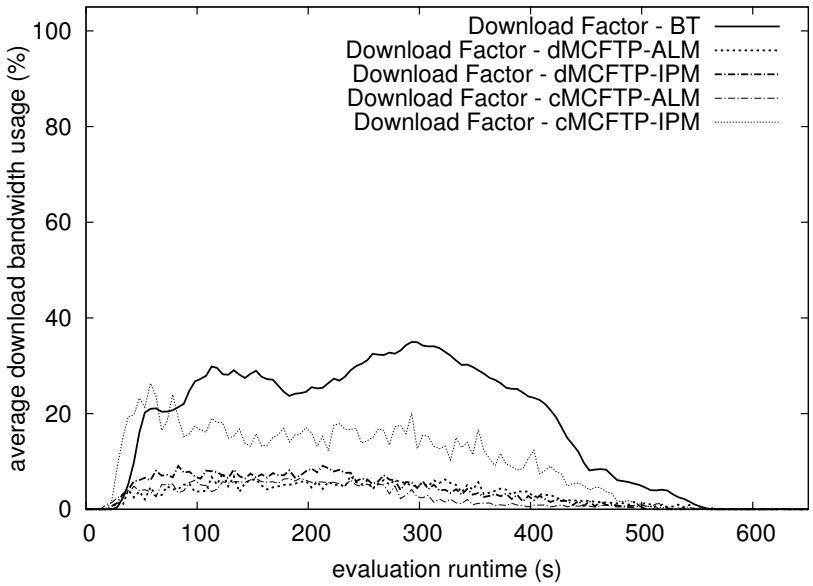


Figure 5.20: Average Download Bandwidth Usage over Time (BitTorrent, dMCFTP, cMCFTP), 8 MB file, 20 Nodes

In Figures 5.20 and 5.21 we compare the average download and upload bandwidth usage over time for BitTorrent, dMCFTP, and cMCFTP with a 8 MB file and 20 Nodes. In Figures 5.22 and 5.23 we compare the average download and upload bandwidth usage over time for BitTorrent, dMCFTP, and cMCFTP with a 8 MB file and 50 Nodes.

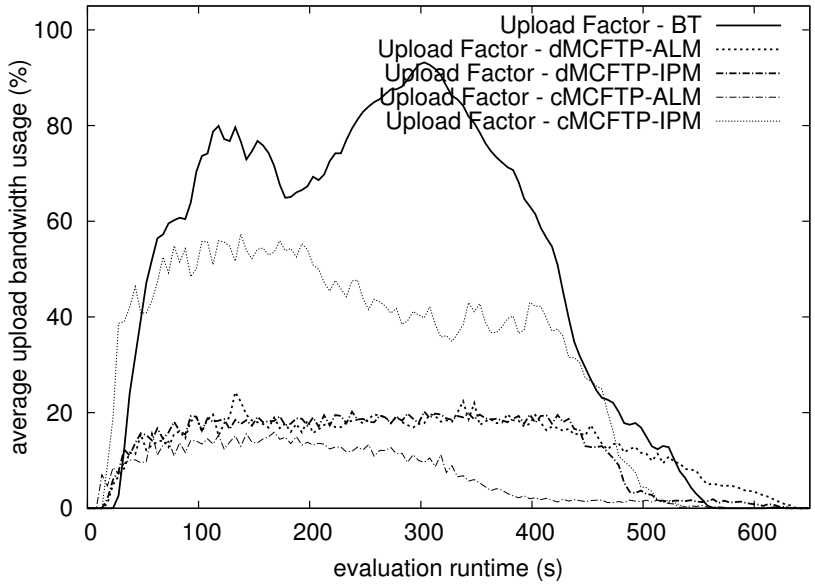


Figure 5.21: Average Upload Bandwidth Usage over Time (BitTorrent, dMCFTP, cMCFTP), 8 MB file, 20 Nodes

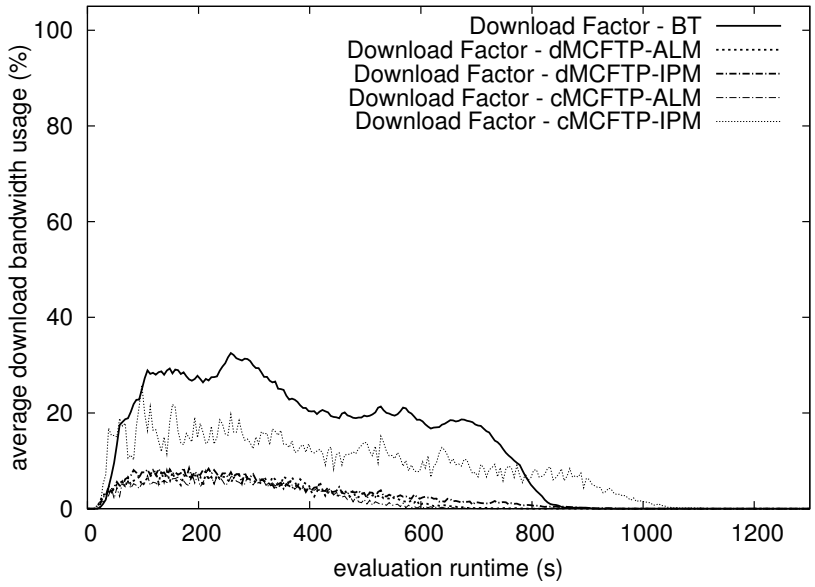


Figure 5.22: Average Download Bandwidth Usage over Time (BitTorrent, dMCFTP, cMCFTP), 8 MB file, 50 Nodes

The small fluctuations in the graphs result from the fact that we built an average of the values over short intervals (rather than plotting a value every second) and also due to the fact that the bandwidth usage can rapidly change depending on

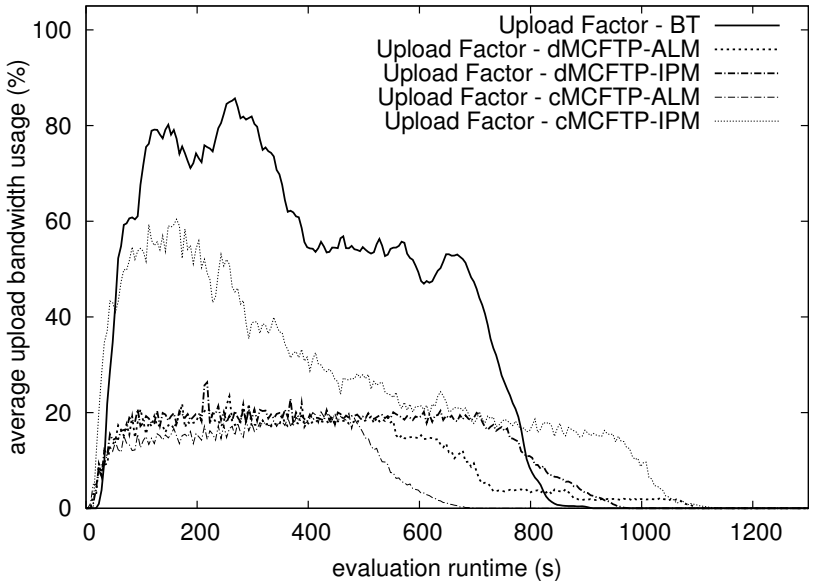


Figure 5.23: Average Upload Bandwidth Usage over Time (BitTorrent, dMCFTP, cMCFTP), 8 MB file, 50 Nodes

chunk availability, SendingGroup timeouts and waiting periods before and after downloading / uploading.

As we can see in Fig. 5.20, BitTorrent generally uses 3–4 times as much download bandwidth of nodes on average than dMCFTP using IP Multicast and ALM. cMCFTP uses roughly half of the download bandwidth at nodes. Therefore, MCFTP is more efficient than BitTorrent regarding resource usage in terms of download bandwidth.

The results for the average download and upload bandwidth usage for 50 nodes as presented in Figures 5.22 and 5.23 are similar to the results when having 20 nodes.

Seeder Development over Time

Finally, we compare the development of seeders over time for different versions of MCFTP with BitTorrent. A node becomes a seeder if it has successfully downloaded the whole file. When all nodes have become seeders, the whole swarm of nodes has finished downloading the file.

Figure 5.24 shows the seeder development in a scenario with 20 nodes downloading a file of 8 MB. After the simulation has run for roughly 300 seconds, all nodes have joined the downloading swarm. The nodes using dMCFTP or cMCFTP with IP Multicast finish earlier than nodes using BitTorrent. After 12 seeders have been established, dMCFTP allows nodes to finish much faster downloading the file compared to using BitTorrent. Please note that in the cMCFTP scenario, the join

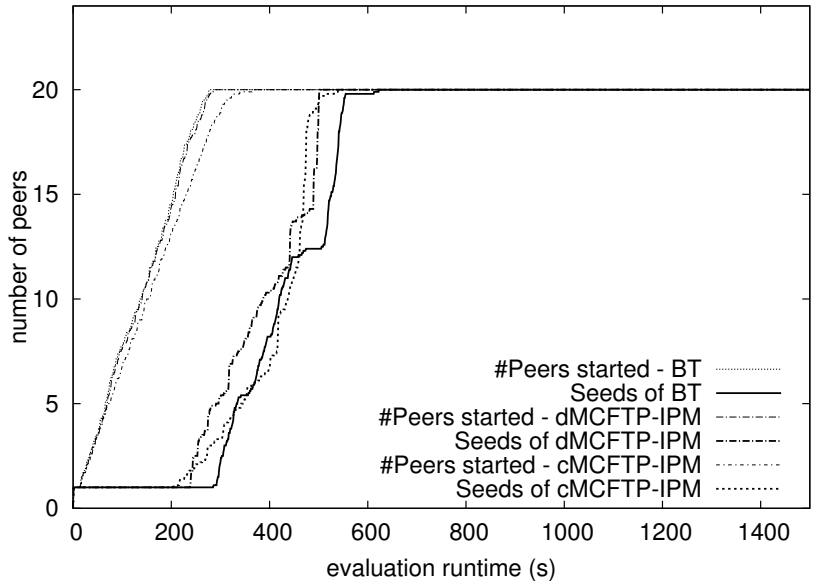


Figure 5.24: Number of Seeders Development over Time (BitTorrent, dMCFTP & cMCFTP with IP Multicast) for 20 Nodes with an 8 MB File

times of nodes are slightly delayed compared to BitTorrent and dMCFTP scenarios. But, cMCFTP still performs better than BitTorrent.

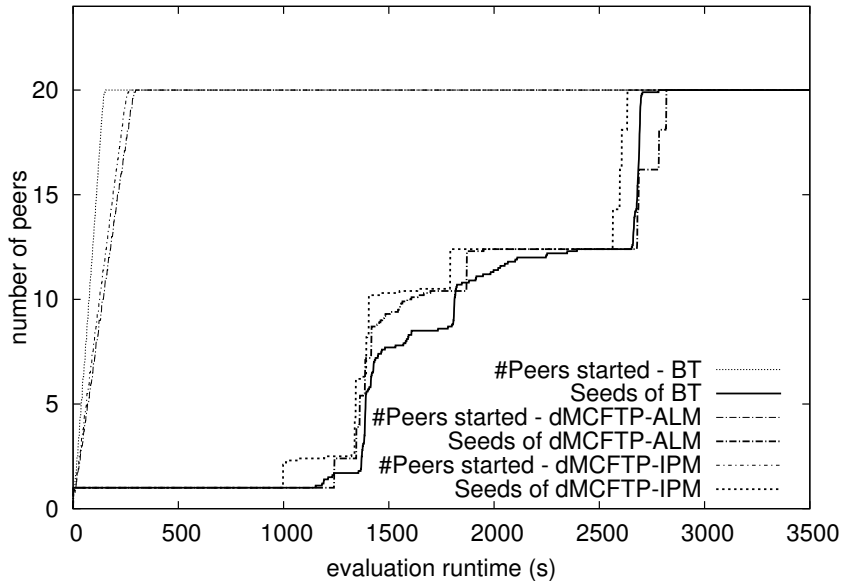


Figure 5.25: Number of Seeders Development over Time (BitTorrent, dMCFTP with ALM and IP Multicast) for 20 Nodes with a 50 MB File

Figure 5.25 presents the seeder development in a scenario with 20 nodes downloading a file of 50 MB. Here, we look at BitTorrent and dMCFTP using IP Multicast as well as ALM. Note that in the BitTorrent scenario, nodes finished joining earlier as for the dMCFTP scenarios. dMCFTP using IP Multicast performs better than BitTorrent during the whole evaluation runtime. When comparing dMCFTP using ALM with BitTorrent, we can see that the last node finishes downloading the file earlier than using BitTorrent. Taking though the delayed joined time of dMCFTP using ALM into account, nodes actually finish their downloads earlier compared to using BitTorrent.

Summary

When comparing our prototype implementation of MCFTP with Azureus, one has to keep in mind that Azureus is a highly optimized version of the BitTorrent protocol that has been under development over many years. Our prototype is though a first proof of concept implementation of the MCFTP protocol for the Internet. Nevertheless, MCFTP already performs better compared to BitTorrent when using IP Multicast. When using MCFTP with ALM, it performs still better for a small number of nodes. For scenarios with more number of nodes, Freepastry is the limiting factor for the performance of MCFTP. But, comparing the average download and upload bandwidth usage, MCFTP is clearly more efficient in terms of usage of a node's resources.

5.5 Conclusion

In this Chapter, we presented MCFTP – a Multicast File Transfer Protocol. There are two modes of MCFTP, the distributed mode called dMCFTP and the centralized mode called cMCFTP. MCFTP can use either IP Multicast or Overlay Multicast.

We compared MCFTP with BitTorrent using the ns2 simulator. Evaluation scenarios for small to large scale networks have been performed using this simulation environment.

Our evaluations using ns2 show that MCFTP using IP Multicast performs better than BitTorrent. In terms of download duration factor, MCFTP performs between 30%-100% better than BitTorrent. Also, using only a few seeders allows MCFTP to perform better than BitTorrent. The differences between cMCFTP and dMCFTP are marginal. dMCFTP scales well for large networks in terms or participating nodes due to its distributed nature. Using cMCFTP with a FileLeader coordinating downloads also scales quite well. In scenario with 2041 nodes, the maximum incoming bandwidth at the FileLeader is between 120–160 Kbps depending on the file size. In the Overlay Multicast scenarios, our evaluations show that MCFTP also performs better than BitTorrent. We analyzed two environments. The first was with QoS support, where parents can always support the bandwidth requirements of all their children. The second did not enforce QoS.

A prototype implementation of MCFTP was compared with Azureus, a popular and very efficient implementation of the BitTorrent protocol. Our prototype implementation supports IP Multicast as well as Overlay Multicast. To support Overlay Multicast, we used Freepastry, which is a freely available implementation of Scribe/Pastry. Unfortunately, the use of Freepastry seems to limit the scalability of the MCFTP protocol when using Overlay Multicast. We performed evaluations with small scale networks for the comparison of MCFTP with BitTorrent.

The evaluations with the prototype implementation also confirm that MCFTP generally performs better than BitTorrent. Especially, MCFTP uses node's resources more efficiently in terms of average upload and download bandwidth usage. Since Azureus is a mature and highly optimized BitTorrent implementation, we are very happy with the results of our proof of concept prototype for MCFTP, which can be still further optimized. Also, another ALM than Scribe/Pastry could be integrated to extend the performance of MCFTP using ALM.

BitTorrent works very well for large scale file distribution scenarios with high churn and failure rates. dMCFTP is also already quite resilient to high churns and node failures due to its completely decentralized and distributed nature. But, also cMCFTP could be enhanced to further improve performance and reliability in such scenarios. The basic implementations of MCFTP used for the evaluations are though not including any mechanisms to support an enhanced behavior/treatment in case of aborted downloads due to nodes leaving or node failures. Some potential extensions to further enhance reliability and to improve support for highly heterogeneous swarms in terms of bandwidth capabilities will be presented in Chapter 6.

Using MCFTP, end users can benefit from an efficient data dissemination mechanism using multicast transmission. Combining this solution with the Multicast Middleware presented in Chapter 4 would enable end users to use MCFTP Internet wide. Finally, adding OM-QoS presented Chapter 3 to the mix would allow end users to benefit from QoS to enhance transmission performance and reliability. We learned that there is still a lot of potential to optimize the MCFTP protocol. Also, changing the ALM used for the prototype will have a big impact on the scalability regarding number of nodes downloading the same file. This should make it even perform better when comparing it to BitTorrent.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

In this thesis, we presented a solution to the question:

> "How can end users benefit from Internet-wide and QoS supporting multicast services to efficiently disseminate data?"

To reach this goal, we proposed three different mechanisms:

- OM-QoS: a Quality of Service for Overlay Multicast framework;
- Multicast Middleware;
- MCFTP: a Multicast Transfer File Protocol.

In detail, we proposed a solution for end users to use Internet-wide IP Multicast services, combined this with a QoS framework, and created an efficient protocol for file distribution, which can benefit from this QoS enabled Internet-wide multicast service.

Therefore, we first introduced in Chapter 3 the OM-QoS framework, which enables QoS for different Peer-to-Peer (P2P) and Application Layer Multicast (ALM) protocols. It facilitates building multicast trees such that they hold certain QoS supporting properties for all multicast paths in an ALM, from the root to leaf nodes. By using the construct of QoS classes, we can map multiple QoS parameters to one discrete value. We described a protocol dependent approach, which modifies P2P protocols, and a protocol independent approach, which is a general solution based on dedicated P2P instances per QoS class.

We applied and evaluated the protocol dependent approach to Scribe, NICE and Chord and the protocol independent approach to CAN. Our evaluations showed that we can ensure QoS for all end-to-end paths in the multicast trees, assuming underlying hard QoS mechanisms, which is difficult to realize with IP Multicast. Furthermore, we can guarantee certain node to root RTT constraints of nodes. The introduced overhead of using OM-QoS in terms of delay, hop-count, fan-out or

node to root RTT is generally at an acceptable level. We learned that it is quite easy to enable QoS for the various P2P/ALM protocols using the protocol dependent and/or independent approach of OM-QoS and the concept of QoS classes. The node to root RTT constraints can though only be met with certain protocols. One precondition for these guarantees is that the receivers need to determine their parents (receiver driven approach) rather than having a forwarder driven multicast implementation.

Furthermore, we presented in Chapter 4 the Multicast Middleware, which allows end users to benefit from a IP Multicast communication service through the Internet, although IP Multicast is not widely deployed. It offers the IP Multicast API to applications and end systems, but uses Overlay Multicast to transport the data with unicast connections. Existing IP Multicast applications do not have to be modified and special infrastructure support does not need to be deployed to the Internet. Our approach also eases QoS support, because IP Multicast traffic is mapped to unicast connections, which can be more easily managed regarding QoS.

We tested and evaluated the Multicast Middleware using multiple scenarios and testing approaches. The evaluations showed that tunneling IP Multicast through an overlay network using ALM is a valid solution in order to provide an Internet-wide IP Multicast service. Capturing and tunneling IP Multicast packets through an ALM introduces only a slight overhead in terms of delay and loss. We are able to support high bandwidth and real-time delay sensitive scenarios such as, multimedia data for video broadcasting, IPTV, VoIP and massively online multiplayer games. We used Scribe/Pastry for setting up the ALM, but this could be easily replaced by another ALM solution.

Finally, we presented in Chapter 5 MCFTP, a Multicast File Transfer Protocol. It allows end users to benefit from multicast distribution mechanisms to exchange and disseminate data efficiently. It uses similar mechanisms as BitTorrent but performs better than BitTorrent as shown by our evaluations using both simulations and a prototype implementation.

MCFTP (Multicast File Transfer Protocol) can operate in two modes, the distributed mode called dMCFTP and the centralized mode called cMCFTP. To transport the data, MCFTP can use either IP Multicast or Overlay Multicast. We compared MCFTP with BitTorrent using the ns2 simulator as well as using a prototype implementation in real testbeds. The evaluations using ns2 and our prototype implementation show that MCFTP generally performs better than BitTorrent. The evaluations in the network simulator ns2 show that dMCFTP scales well for large network sizes. Using our prototype implementation, we demonstrated that MCFTP is also more efficient regarding resource utilization of nodes in terms of average upload and download bandwidth usage. End users can download their requested files faster while using less upload and download bandwidth on their end systems.

By combining these three mechanisms, end users can finally profit from efficient data transmission and dissemination using QoS enabled multicast services Internet-wide.

6.2 Outlook

The OM-QoS framework has been applied to different protocols using both approaches, the protocol dependent as well as the protocol independent approach. Since it is quite easy to integrate other P2P/ALM protocols into the OM-QoS framework, suitable protocols such as Bayeux/Tapestry, VRing or Borg could be implemented in OMNet++ with our framework. These protocols could then be analyzed as well and compared with the existing implemented protocols. This would then help to identify the best possible match of P2P/ALM QoS combination for certain application scenarios, such as multimedia-streaming, networked virtual environments, VoIP, etc. As a further step, the presented OM-QoS mechanisms for Scribe, NICE, Chord and CAN could be implemented in prototype implementations to be actually used in the Internet. These could then be evaluated using real network environments, either in local networks or testbeds in distributed systems such as PlanetLab.

We also started to investigate how reputation based systems could help to improve QoS and QoE for end users participating in P2P/ALM networks. These reputation reports are distributed using gossiping mechanisms [66]. Therefore, we propose REPOM (Reputation Based Overlay Multicast) [47], which identifies non-cooperative or selfish nodes in a distributed manner. Nodes can then decide by themselves according to the gossiped reports they received if they have to find another, more suitable and better performing parent node for the multicast data subscription.

Multicast could also be efficiently used for data dissemination in Wireless Sensor Networks (WSNs), but should support QoS in terms of reliability for critical tasks such as code updates. We proposed a solution for designing a reliable multicast solution based on IP Multicast and Overlay Multicast in [175]. This proposed solution could be further enhanced with OM-QoS mechanisms to enable QoS-aware multicast tree construction in WSNs.

Further enhancements of OM-QoS could include adaptive re-encoding of multimedia data on nodes acting as mediators between different QoS classes. Hence, if a node having a higher QoS class has to transmit, e.g., a video stream using a certain average bandwidth to a node with a lower QoS class (not supporting the average bandwidth of the currently used stream format), then the node with the higher QoS class could re-encode the stream before forwarding it. Also, transcoding / scalable coding [85] could be combined with OM-QoS and its QoS class construct. The video stream could be split into multiple trans-coded layers, which would be only forwarded fully or partially depending on the QoS class of a node. If a node has to forward the multicast data to a node having the same QoS class, then all trans-coded layers received could also be forwarded. But if the child node, which has to be served with forwarded data only has a lower QoS class, some of those layers could be omitted from being forwarded to that node.

Finally, more and more P2P/ALM concepts and applications are emerging that are strongly aware about their underlying network infrastructure. They could also

profit heavily from OM-QoS mechanisms to enable building QoS-aware multicast trees that are actually supported by the underlying network infrastructure in terms of QoS. Many existing third generation P2P networks and projects, such as 3GP2P [176], which are facing the problems of high member churn or member selfishness could also benefit from or cooperate with OM-QoS mechanisms by incorporating / combining these aspects with the presented QoS class construct.

The Multicast Middleware has been thoroughly tested in local test beds regarding performance. Other tests not focusing on performance regarding throughput but rather concerning delays could also be performed in distributed network testbeds, such as PlanetLab. This would give us additional insights regarding overhead in networks with large delays and also with limited resources regarding bandwidth and computing power due to the shared nature of PlanetLab nodes and environments. Also, other P2P/ALM protocols besides Scribe/Pastry could be integrated and evaluated with the Multicast Middleware. The interface to the ALM implementation used in the Multicast Middleware was developed in a general and ALM independent way. Since we use an optimized serialization mechanism and our own protocol to transmit the data, using another ALM would mainly impact the distribution tree and overlay topology. Therefore, depending on the application scenario, the appropriate P2P/ALM scheme delivering the best performance could be used.

MCFTP has been evaluated using simulations and with a first prototype implementation in real network testbeds. Regarding the prototype implementation comparison with Azureus, we learned that we still can improve the performance of MCFTP. Therefore, we should investigate further strategies for either the chunk selection performed by the FileLeader or by nodes for the distributed MCFTP approach. These strategies could also be adaptive, e.g., using different optimizations for small and large networks. Taking into account the number of seeders or the lifetime of a swarm / nodes could help to improve the overall and also actual performance. Also, trying to support asymmetric bandwidth capabilities of nodes (in terms of maximum possible upload and download rate) more uniformly as presented in [149] could be beneficial to improve overall or node specific performance. MCFTP can also be further optimized to work reliably in large-scale scenarios with heterogeneous hosts and high churn / failure rates. Such extensions include multiple levels of FileManagementGroups, where hosts are grouped according their bandwidth capabilities. This would further improve scalability and also consolidate heterogeneous bandwidth capabilities leading to more potential providers of chunks matching the bandwidth capabilities of a downloading node. Furthermore, supporting on-the fly switching to other chunk providers (i.e., other SendingGroups for the same chunk that are active at the same time) while downloading a chunk would further increase reliability regarding high churn and failure rates. The prototype implementation could be enhanced by supporting fall back and recovery mechanisms when a FileLeader in cMCFTP would suddenly disappear. This would include providing backup FileLeaders as well as negotiation and handover mechanisms to determine new FileLeaders in case of failures as proposed in [128]. Other extensions could include self-controlling and self-healing mecha-

nisms to detect/avoid malicious behavior. Malicious nodes could be modeled and counter-measurements for free-riding or sabotaging nodes could be designed, integrated and evaluated. Additionally, security and anonymity extensions would be an additional benefit to MCFTP. For cMCFTP, the message transfer from nodes to the FileLeader could be encrypted by using public/private key mechanisms. Extensions supporting mutual anonymity [181] for senders and receivers could be integrated when using Overlay Multicast with MCFTP, but this would also introduce additional computation and communication overhead.

Integrating REPOM [47] mechanisms could also help to enforce cooperation and build a similar strategy as the tit-for-that mechanism in BitTorrent. As a final step, MCFTP could be evaluated in distributed testbed environments such as PlanetLab. Due to the limited processing power of PlanetLab nodes and their shared usage nature, the existing proof-of-concept prototype written in Java and using Freepastry is not the optimal candidate for this task. Also, the problem of deploying the experiment and collecting the results in large scale networks with a few hundred nodes is not feasible with the current solution package used for the prototype evaluation. Therefore, a better solution would be to implement MCFTP using SPLAY [102]. It facilitates development of network protocols like MCFTP and eases deployment and evaluation on a large scale. Also, a BitTorrent implementation for SPLAY already exists, which makes it easier to compare it regarding performance.

Finally, combining a mature version of MCFTP with an integrated, self-installing Multicast Middleware and adding QoS capabilities using OM-QoS to MCFTP would enable end users to be able to use MCFTP Internet wide while benefiting from QoS to enhance transmission performance and reliability.

Chapter 7

Acronyms

ALM	Application Layer Multicast
ALTO	Application-Layer Traffic Optimization
API	Application Programming Interface
AS	Autonomous System
A/V	Audio / Video
BGP	Border Gateway Protocol
BIOS	Basic Input / Output System
BRITE	Boston University Representative Internet Topology Generator
CAN	Content Addressable Networks
CDN	Content Distribution Network
CERNET	China Education and Research Network
COPS	Common Open Policy Service
CPU	Central Processing Unit
DDR	Double Data Rate
DiffServ	Differentiated Services
DNS	Domain Name System
DHT	Distributed Hash Table
DoS	Denial of Service
DSL	Digital Subscriber Line

ESM	End System Multicast
EuQoS	End-to-end QoS Support over Heterogeneous Networks
FL	File Leader
FMG	File Management Group
FTP	File Transfer Protocol
IANA	Internet Assigned Numbers Authority
ID	Identifier
IGMP	Internet Group Management Protocol
IntServ	Integrated Services
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
IVS	INRIA Videoconferencing System
I/O	Input / Output
KA	Keep Alive
LAN	Local Area Network
LMSG	Live Media Service Grid
LSSR	Loose Source Routing
HTTP	Hypertext Transfer Protocol
MAST	Multicast Application Sharing Tool
MBONE	Multicast Backbone
MCFTP	Multicast File Transfer Protocol
MGEN	Multi Generator
MM-VISA	Massively Multiuser VIrtual Simulation Architecture
MPEG	Moving Picture Experts Group
MMOG	Massively Multiplayer Online Game
NAT	Network Address Translator

NeVoT	Network Voice Terminal
NV	Network Video Tool
NICE	NICE is the Internet Cooperative Environment
OM-QoS	Quality of Service for Overlay Multicast
P2P	Peer-to-Peer
PC	Personal Computer
QIOM	QoS-satisfied Inter-domain Overlay Multicast
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RNG	Random Number Generator
RSVP	Resource Reservation Protocol
RTT	Round Trip Time
SAP	Session Announcement Protocol
SDP	Session Description Protocol
SG	Sending Group
SQL	Structured Query Language
SvB	Service Brokers Node
TAG	Topology Aware Grouping
TCP	Transport Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UPNP	Universal Plug and Play
VAT	Visual Audio Tool
VIC	Video Conferencing Tool
VLC	Video LAN Client
VoD	Video on Demand

VoIP	Voice over Internet Protocol
VON	Voronoi-based Overlay Network
WB	White Board
WG	Working Group

Bibliography

[1] "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," Internet Engineering Task Force, RFC 2205, Sept. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2205.txt

[2] "The COPS (Common Open Policy Service) Protocol," Internet Engineering Task Force, RFC 2748, Jan. 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2748.txt

[3] "Adaptive Routing of QoS-Constrained Media Streams over Scalable Overlay Topologies," in *RTAS '04: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2004, p. 518.

[4] S. Agarwal, J. P. Singh, A. Mavlankar, P. Baccichet, and B. Girod, "Performance and Quality-of-Service Analysis of a Live P2P Video Multicast Session on the Internet," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, 2008, pp. 11–19. [Online]. Available: http://dx.doi.org/10.1109/IWQOS.2008.7

[5] D. T. Ahmed, S. Shirmohammadi, and J. C. Oliveira, "A hybrid P2P communications architecture for zonal MMOGs," *Multimedia Tools Appl.*, vol. 45, no. 1-3, pp. 313–345, 2009.

[6] Z. Albanna, K. Almeroth, D. Meyer, and M. Schipper, "IANA Guidelines for IPv4 Multicast Address Assignments," Internet Engineering Task Force, RFC 3171, Aug. 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3171.txt

[7] W. Allcock, J. Bester, J. Bresnahan, S. Meder, P. Plaszczak, and S. Tuecke, "GridFTP: Protocol Extensions to FTP for the Grid, Proposed Recommendation GFD-R-P.020, Global Grid Forum," April 2003.

[8] "Application-Layer Traffic Optimization (ALTO) Work Group," 2010. [Online]. Available: http://www.ietf.org/html.charters/alto-charter.html

[9] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 131–145, 2001.

[10] S. Androutsellis-Theotokis and D. Spinellis, "A survey of Peer-to-Peer content distribution technologies," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 335–371, 2004. [Online]. Available: http://doi.acm.org/10.1145/1041680.1041681

[11] E. Angori, G. Martufi, M. Brogle, D. Milic, *et al.*, "D1.1.2: System Design: Functions, Interfaces Specification," Tech. Rep., May 2005.

[12] "Azureus Sourceforge Website," 2010. [Online]. Available: http://azureus.sourceforge.net/

[13] F. Baker, C. Iturralde, F. Faucheur, and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 Reservations," Internet Engineering Task Force, RFC 3175, Sept. 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3175.txt

[14] R. Balmer and T. Braun:, "Resource Control and Authentication for a Video Streaming Service in a DiffServ/IP Multicast Network," in *3rd Conference on Security and Network Architectures (SAR04)*, La Londe, Cote d'Azur France, June 21-25 2004.

[15] S. Banerjee, B. Bhattacharjee, and K. Christopher, "Scalable Application Layer Multicast," in *SIGCOMM02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 32, no. 4. New York,USA: ACM, 2002, pp. 205–217. [Online]. Available: http://portal.acm.org/citation.cfm?id=633045

[16] S. Barthlomé, "Quality of Service for Overlay Multicast applied to the NICE Protocol," Bern, Switzerland, September 2009.

[17] D. Bauer, I. Iliadis, S. Rooney, and P. Scotton, "Communication Architectures for Massive Multi-Player Games," *Multimedia Tools Appl.*, vol. 23, no. 1, pp. 47–66, 2004.

[18] D. Bauer and S. Rooney, "The Performance of Software Multicast-Reflector Implementations for Multi-player Online Games," in *Networked Group Communication*, 2003, pp. 214–225.

[19] L. Bettosini, "Performance Comparison of Native Multicast versus Overlay Multicast," Bern, Switzerland, April 2008.

[20] ——, "Quality of Service for Overlay Multicast Content Addressable Network (CAN)," Bern, Switzerland, August 2009.

[21] E. W. Biersack, P. Rodriguez, and P. Felber, "Performance Analysis of Peer-to-Peer Networks for File Distribution," in *In Proc. Fifth International Workshop on Quality of Future Internet Services (QofIS'04*, 2004.

[22] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," Internet Engineering Task Force, RFC 2475, Dec. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2475.txt

[23] R. Bless and K. Wehrle, "IP Multicast in Differentiated Services (DS) Networks," Internet Engineering Task Force, RFC 3754, Apr. 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3754.txt

[24] A. Bozdog, R. van Renesse, and D. Dumitriu, "SelectCast: a scalable and self-repairing multicast overlay routing facility," in *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*. New York, NY, USA: ACM Press, 2003, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/1036921.1036925

[25] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," Internet Engineering Task Force, RFC 1633, June 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1633.txt

[26] Bram Cohen, "Incentives Build Robustness in BitTorrent," *In Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, May 2003.

[27] T. Braun, V. Arya, and T. Turletti, "Explicit routing in multicast overlay networks," *Computer Communications*, vol. 29, no. 12, pp. 2201–2216, August 2006.

[28] T. Braun, M. Diaz, J. Enrquez-Gabeiras, and T. Staub, *End-to-End Quality of Service Over Heterogeneous Networks*. Springer, 2008.

[29] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in Network Simulation," *Computer*, vol. 33, no. 5, pp. 59–67, 2000.

[30] "BRITE - Boston University Representative Internet Topology gEnerator," 2010. [Online]. Available: http://www.cs.bu.edu/brite/

[31] M. Brogle, "OM-QoS: Overlay Multicast Quality of Service," Institute of Computer Science and Applied Mathematics, Bern, Tech. Rep. IAM-07-004, December 2007, in "RVS Retreat 2007 at Quarten", Quarten, Switzerland.

[32] ——, "Reviving IP Multicast Using QoS enhanced Overlay Networks," Institute of Computer Science and Applied Mathematics, Bern, Tech. Rep. IAM-08-003, November 2008, in "BeNeFri Summer School 2008 on Dependable Systems", Münchenwiler, Switzerland.

[33] ——, "Quality of Service for "NICE" Overlay Multicasting,," Institute of Computer Science and Applied Mathematics, Bern, Tech. Rep. IAM-09-006, September 2009, in "BeNeFri Summer School 2009 on Dependable Systems", Münchenwiler, Switzerland.

[34] M. Brogle, S. Barthlomé, and T. Braun, "Quality of Service for Multicasting using NICE," in *25th Symposium On Applied Computing (ACM SAC 2010)*. ACM, March 2010.

[35] M. Brogle, L. Bettosini, and T. Braun, "Quality of Service for Multicasting in Content Addressable Networks," in *12th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS 09)*. Springer LNCS 5842, October 2009, pp. 170–175.

[36] M. Brogle, A. Geycasar, and T. Braun, "MCFTP – Implementation and Evaluation of a Multicast File Transfer Protocol," to be submitted for publication.

[37] M. Brogle and D. Milic, "EuQoS Multicast Midddleware: Basic Architecture Overview and Concepts," Institute of Computer Science and Applied Mathematics, Bern, Tech. Rep. IAM-05-002, June 2005, in "Retreat of the Computer Networks and Distributed Systems research group at Griesalp", Kiental, Switzerland.

[38] ——, "Multicast Middleware Implementation," 2010. [Online]. Available: http://www.iam.unibe.ch/%7Ervs/research/euqos/mcast-4.0.4_r701.zip

[39] ——, "Multicast Middleware Installation Manual," 2010. [Online]. Available: http://www.iam.unibe.ch/%7Ervs/research/euqos/mcast-4.0.4_r701.zip

[40] M. Brogle, D. Milic, L. Bettosini, and T. Braun, "A Performance Comparison of Native IP Multicast and IP Multicast Tunneled through a Peer-to-Peer Overlay Network," in *2009 International Workshop on Peer-To-Peer Networking (P2PNet'09) in conjunction with ICUMT 2009*. IEEE, October 2009.

[41] M. Brogle, D. Milic, and T. Braun, "QoS Enabled Multicast for Structured P2P Networks," in *Workshop on Peer-to-Peer Multicasting at the 4th IEEE Consumer Communications and Networking Conference*. IEEE, January 2007, pp. 991–995.

[42] ——, "Supporting IP Multicast Streaming Using Overlay Networks," in *QShine: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Vancouver, British Columbia, Canada,, August 14 - 17 2007.

[43] ——, "Quality of Service for Peer-to-Peer based Networked Virtual Environments," in *P2P-NVE 2008 Workshop at the 14th IEEE International Conference on Parallel and Distributed Systems*. Melbourne, Victoria, Australia: IEEE, December 2008, pp. 847–852.

[44] ——, "Multicast-Middleware der Universität Bern, Schweiz, resultierend aus EU FP6 IST IP "EuQoS" - Gewinner des "KuVS Communication Software Preis" 2009," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 32, no. 3, pp. 176–178, 10 2009.

[45] M. Brogle, D. Paprtitz, D. Milic, and T. Braun, "MCFTP - A Multicast File Transfer Protocol for Efficient Data Dissemination," to be submitted for publication.

[46] M. Brogle, A. Rüttimann, and T. Braun, "Quality of Service for Overlay Multicasting in Chord," in *The Fifteenth IEEE Symposium on Computers and Communications (ISCCâ10)*. IEEE, 2010 2010.

[47] U. Bürgi, "REPOM: Reputation Based Overlay Multicast," Bern, Switzerland, December 2009.

[48] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," Internet Engineering Task Force, RFC 3376, Oct. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3376.txt

[49] M. Castro, M. B. Jones, A. M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An Evaluation of Scalable Application-Level Multicast built using Peer-to-Peer Overlays," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, 2003, pp. 1510–1520 vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1208986

[50] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, New York, NY, USA, October 2003, pp. 298–313. [Online]. Available: http://doi.acm.org/10.1145/945445.945474

[51] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "One Ring to rule them all: Service Discover and Binding in Structured Peer-to-Peer Overlay Networks," in *SIGOPS European Workshop*, Sep 2002.

[52] ——, "Scribe: a Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, no. 8, pp. 1489–1499, October 2002.

[53] L. Cherkasova and J. Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks," in *Proceedings of USITS '03: 4th USENIX Symposium on Internet Technologies and Systems*. USENIX Association, March 26-28 2003, pp. 85 – 98.

[54] "Chord/DHash Project," 2010. [Online]. Available: http://pdos.csail.mit.edu/chord/

[55] "Chordless," 2010. [Online]. Available: http://chordless.sourceforge.net/

[56] "Conference XP," 2010. [Online]. Available: http://www.codeplex.com/ConferenceXP

[57] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "QoS's downfall: at the bottom, or not at all!" in *RIPQoS '03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*. New York, NY, USA: ACM Press, 2003, pp. 109–114. [Online]. Available: http://dx.doi.org/10.1145/944592.944594

[58] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[59] Y. Dan, C. Xinmeng, and C. Yunlei, "An Improved P2P Model Based on Chord," in *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, 2005, pp. 807–811.

[60] S. Deering, "Host extensions for IP multicasting," Internet Engineering Task Force, RFC 1112, Aug. 1989. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1112.txt

[61] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *Network, IEEE*, vol. 14, no. 1, pp. 78 –88, jan/feb 2000.

[62] S. Duarte, L. J. Martins, H. J. Domingos, and N. Preguiça, "A case study on event dissemination in an active overlay network environment," in *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*. New York, NY, USA: ACM Press, 2003, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1145/966618.966624

[63] A. El-Sayed, V. Roca, and L. Mathy, "A survey of proposals for an alternative group communication service," *Network, IEEE*, vol. 17, no. 1, pp. 46 – 51, jan/feb 2003.

[64] H. Eriksson, "MBONE: the multicast backbone," *Commun. ACM*, vol. 37, no. 8, pp. 54–60, 1994. [Online]. Available: http://doi.acm.org/10.1145/179606.179627

[65] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, "Network Visualization with Nam, the VINT Network Animator," *Computer*, vol. 33, no. 11, pp. 63–68, 2000.

[66] P. Eugster, P. Felber, and F. Le Fessant, "The "art" of programming gossip-based systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 37–42, 2007.

[67] "End to End Quality of Service over Heterogeneous Networks – A White Paper from the EuQoS Consortium," 2004. [Online]. Available: http://ec.europa.eu/information_society/istevent/2006/cf/document.cfm?doc_id=616

[68] "EuQoS project website," 2010. [Online]. Available: http://www.euqos.eu

[69] S. Fahmy and M. Kwon, "Characterizing overlay multicast networks," in *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, 2003.

[70] K. Fall, "Network Emulation in the Vint/NS Simulator," in *ISCC '99: Proceedings of the The Fourth IEEE Symposium on Computers and Communications*. Washington, DC, USA: IEEE Computer Society, 1999, p. 244.

[71] B. Fenner and et. al, "mrouted 3.9-beta," 2010. [Online]. Available: ftp://ftp.parc.xerox.com/pub/net-research/ipmulti/

[72] W. Fenner, "Internet Group Management Protocol, Version 2," Internet Engineering Task Force, RFC 2236, Nov. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2236.txt

[73] R. Finlayson, "IP Multicast and Firewalls," Internet Engineering Task Force, RFC 2588, May 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2588.txt

[74] "Fit in IT – Wanderausstellung im Rahmen des Föderprogramms FIT der Hasler Stiftung," 2010. [Online]. Available: http://www.fit-in-it.ch/

[75] "Freepastry," 2010. [Online]. Available: http://freepastry.rice.edu/

[76] A. Ganjam and H. Zhang, "Connectivity restrictions in overlay multicast," in *Proceedings of the 14th ACM international workshop on Network and operating systems support for digital audio and video (NOSSDAV '04)*, New York, 2004, pp. 54–59. [Online]. Available: http://doi.acm.org/10.1145/1005847.1005860

[77] A. Geycasar, "MC-FTP (Multicast File Transfer Protocol): Implementation and Comparison with BitTorrent," Master's thesis, University of Bern, Bern, Switzerland, 2010.

[78] R. Ghosh and G. Varghese, "Congestion Control in Multicast Transport Protocols," Washington University in St. Louis, USA, Tech. Rep., June 1998.

[79] "The Globus Alliance," 2010. [Online]. Available: http://www.globus.org/

[80] J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, ser. Java Series. Sun Microsystems, 1996.

[81] "GridFTP Working Group," 2010. [Online]. Available: http://forge.gridforum.org/projects/gridftp-wg/

[82] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang, "A Tree-based Forward Digest Protocol to Verify Data Integrity in Distributed Media Streaming," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, July 2005.

[83] M. Handley and V. Jacobson, "SDP: Session Description Protocol," Internet Engineering Task Force, RFC 2327, Apr. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2327.txt

[84] M. Handley, C. Perkins, and E. Whelan, "Session Announcement Protocol," Internet Engineering Task Force, RFC 2974, Oct. 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2974.txt

[85] U. Horn, K. Stuhlmüller, M. Link, and B. Girod, "Robust Internet video transmission based on scalable coding and unequal error protection," *Signal Processing: Image Communication*, vol. 15, no. 1-2, pp. 77 – 94, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/B6V08-3XJKMYT-6/2/5c75dc4571d386ee3037bcc4f95acfa1

[86] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A Survey of Application-Layer Multicast Protocols," *Communications Surveys & Tutorials, IEEE*, vol. 9, no. 3, pp. 58–74, 2007.

[87] M. Hosseini and N. D. Georganas, "End System Multicast Protocol for Collaborative Virtual Environments," *Presence: Teleoper. Virtual Environ.*, vol. 13, no. 3, pp. 263–278, 2004.

[88] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 1134–1138.

[89] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "VON: a scalable Peer-to-Peer Network for Virtual Environments," *Network, IEEE*, vol. 20, no. 4, pp. 22–31, August 2006.

[90] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, vol. 20, no. 8, 2002.

[91] "Internet Multicast Addresses," 2010. [Online]. Available: http://www.iana.org/assignments/multicast-addresses/

[92] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 2008.

[93] "INRIA Videoconferencing System (ivs)," 2010. [Online]. Available: http://www.inria.fr/rodeo/ivs.html

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

[95] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable Multicasting with an Overlay Network," pp. 197–212. [Online]. Available: citeseer.ist.psu.edu/jannotti00overcast.html

[96] JTC1/SC29/WG11, "ISO/IEC 11172:1993: Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Parts 1 to 5," 1993.

[97] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM Press, 1997, pp. 654–663. [Online]. Available: http://dx.doi.org/10.1145/258533.258660

[98] S. Kiesel, L. Popkin, S. Previdi, R. Woundy, and Y. R. Yang, "ALTO Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-alto-protocol-03.txt, March 2010, informational. [Online]. Available: http://www.ietf.org/id/draft-ietf-alto-protocol-03.txt

[99] ——, "Application-Layer Traffic Optimization (ALTO) Requirements," Internet Engineering Task Force, Internet-Draft draft-ietf-alto-reqs-04, March 2010, informational. [Online]. Available: http://www.ietf.org/id/draft-ietf-alto-reqs-04.txt

[100] D. Kostic;, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *SOSP '03: Proceedings*

of the nineteenth ACM symposium on Operating systems principles. New York, NY, USA: ACM Press, 2003, pp. 282–297.

[101] M. Kwon and S. Fahmy, "Path-aware overlay multicast," *Comput. Networks*, vol. 47, no. 1, pp. 23–45, January 2005. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2004.06.025

[102] L. Leonini, E. Rivière, and P. Felber, "SPLAY: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze)," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2009, pp. 185–198.

[103] G. Lewis, S. M. Hasan, V. N. Alexandrov, and M. T. Dove, "Facilitating Collaboration and Application Sharing with MAST and the Access Grid Development Infrastructures," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2006, p. 68.

[104] D. Li, J. Wu, Y. Cui, and J. Liu, "QoS-Aware Streaming in Overlay Multicast Considering the Selfishness in Construction Action," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, May 2007, pp. 1154–1162. [Online]. Available: http://dx.doi.org/10.1109/INFCOM.2007.138

[105] Z. Li, "Resiliency and Quality-of-Service (QoS) support in multicasting and overlay networks," Ph.D. dissertation, Davis, CA, USA, 2005, adviser-Prasant Mohaptra.

[106] Z. Li and P. Mohapatra, "HostCast: a new overlay multicasting protocol," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 1, June 2003, pp. 702–706 vol.1. [Online]. Available: http://dx.doi.org/10.1109/ICC.2003.1204265

[107] ——, "QRON: QoS-aware routing in overlay networks," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 29–40, 2004.

[108] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *Communications Surveys & Tutorials, IEEE*, pp. 72–93, 2005.

[109] H. Ma and K. G. Shin, "Multicast Video-on-Demand services," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 1, pp. 31–43, 2002.

[110] "Multicast Application Sharing Tool (MAST)," 2010. [Online]. Available: http://acet.reading.ac.uk/projects/mast/index.php

[111] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[112] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 346.

[113] "Multi-Generator (MGEN)," 2010. [Online]. Available: http://cs.itd.nrl.navy.mil/work/mgen/

[114] D. Milic, M. Brogle, and T. Braun, "Video Broadcasting using Overlay Multicast," in *ISM '05: Proceedings of the Seventh IEEE International Symposium on Multimedia*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 515–522.

[115] C. K. Miller, *Multicast Networking and Applications*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[116] E. Mingozzi, G. Stea, M. Callejo-Rodríguez, J. Enríquez-Gabeiras, G. G. de Blas, F. Ramón-Salquero, W. Burakowski, A. Beben, J. Sliwinski, H. Tarasiuk, O. Dugeon, M. Diaz, L. Baresse, and E. Monteiro, "EuQoS: End-to-End Quality of Service over Heterogeneous Networks," *Computer Communications*, vol. 32, no. 12, pp. 1355 – 1370, 2009, special Issue of Computer Communications on Heterogeneous Networking for Quality, Reliability, Security, and Robustness - Part II.

[117] "MOMOCOMM: Multiobjective Optimized Multicast Communication Primitive for P2P Collaboration," 2010. [Online]. Available: http://code.google.com/p/momocomm/

[118] M. Mysore and G. Varghese, "FTP-M: An FTP-like Multicast File Transfer Application," University of California at San Diego, La Jolla, CA, USA, Tech. Rep., 2001.

[119] "Network simulator ns-2," 2010. [Online]. Available: http://www.isi.edu/nsnam/ns/

[120] "Guide to NeVoT 3.34," 2010. [Online]. Available: http://www.inria.fr/rodeo/ivs.html

[121] "NeVoT 3.36 Source Code," 2010. [Online]. Available: http://www.cs.columbia.edu/irt/software/nevot/source/

[122] M. Nikolic, "WinJTAP Interface for Multicast Middleware on the Win32 Platform," Bern, Switzerland, March 2008.

[123] "nv - Network video tool," 2010. [Online]. Available: ftp://ftp.parc.xerox.com/pub/net-research/nv-3.3beta

[124] "OMNET++ community site," 2010. [Online]. Available: http://www.omnetpp.org

[125] "Open Chord," 2010. [Online]. Available: http://open-chord.sourceforge.net/

[126] "Open Grid Forum," 2010. [Online]. Available: http://www.gridforum.org/

[127] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Supporting Heterogeneity and Congestion Control in Peer-to-Peer Multicast Streaming," in *IPTPS*, 2004, pp. 54–63.

[128] D. Papritz, "MC-FTP (Multicast File Transfer Protocol): Simulation and Comparison with BitTorrent," Master's thesis, University of Bern, Bern, Switzerland, 2010.

[129] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, 2003.

[130] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building PlanetLab," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 351–366.

[131] "PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services," 2010. [Online]. Available: http://planet-lab.org/

[132] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA'97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 1997, pp. 311–320. [Online]. Available: http://doi.acm.org/10.1145/258492.258523

[133] J. Postel, "Internet Protocol," Internet Engineering Task Force, RFC 791, Sept. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt

[134] ——, "Transmission Control Protocol," Internet Engineering Task Force, RFC 793, Sept. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[135] J. Postel and J. Reynolds, "File Transfer Protocol," Internet Engineering Task Force, RFC 959, Oct. 1985. [Online]. Available: http://www.rfc-editor.org/rfc/rfc959.txt

[136] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. ACM Press, 2001, pp. 161–172. [Online]. Available: http://portal.acm.org/citation.cfm?id=383072

[137] S. Ratnasamy, M. Handley, R. Karp, and S. Schenker, "Application-Level Multicast Using Content-Addressable Networks," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. London, UK: Springer-Verlag, 2001, pp. 14–29. [Online]. Available: http://portal.acm.org/citation.cfm?id=648089.747491

[138] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Network Working Group, RFC 4271, January 2006, Obsoletes: RFC 1771. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4271.txt

[139] "Resilient Overlay Networks," 2010. [Online]. Available: http://nms.lcs.mit.edu/ron/

[140] P. Rizk, C. Kiddle, R. Simmonds, and B. Unger, "Performance of a GridFTP overlay network," *Future Gener. Comput. Syst.*, vol. 24, no. 5, pp. 442–451, 2008.

[141] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, 1997.

[142] B. G. Rocha, V. Almeida, and D. Guedes, "Increasing QoS in Selfish Overlay Networks," *IEEE IC*, vol. 10, no. 3, pp. 24–31, 2006. [Online]. Available: http://dx.doi.org/10.1109/MIC.2006.54

[143] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329–350.

[144] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Networked Group Communication, Third International COST264 Workshop (NGC'2001)*, ser. Lecture Notes in Computer Science, vol. 2233, Nov. 2001, pp. 30–43.

[145] A. Rüttimann, "Quality of Service, End to End Delays and Overlay Multicast for Structured P2P Networks like Chord," Master's thesis, University of Bern, Switzerland, Bern, Switzerland, 2009.

[146] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of Internet content delivery systems," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 315–327, 2002.

[147] K. Savetz, N. Randall, and Y. Lepage, *MBONE: Multicasting Tomorrow's Internet*. Foster City, CA, USA: IDG Books Worldwide, Inc., 1995.

[148] M. Scheidegger, T. Braun, and F. Baumgartner, "Endpoint Cluster Identification for End-to-End Distance Estimation," in *International Conference on Communications, Istanbul, Turkey*. IEEE, June 2006, CD-ROM.

[149] M. Schiely and P. Felber, "Peer-to-Peer Distribution Architectures Providing Uniform Download Rates," in *International Symposium on Distributed Objects and Applications (DOA'05), Agia Napa, Cyprus*, ser. Lecture Notes in Computer Science, vol. 3761. Springer, October 2005, pp. 1083–1096.

[150] C. Schill, "SMCRoute," 2010. [Online]. Available: http://www.cschill.de/smcroute/

[151] H. Schulzrinne, "Voice Communication Across the Internet: A Network Voice Terminal," Department of Computer Science, University of Massachusetts, Tech. Rep., July 1992.

[152] J. Seedorf and E. Burger, "Application-Layer Traffic Optimization (ALTO) Problem Statement," Internet Engineering Task Force, RFC 5693, October 2009. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5693.txt

[153] S. Shalunov and B. Teitelbaum, "Quality of Service and Denial of Service," in *RIPQoS '03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*. New York, NY, USA: ACM Press, 2003, pp. 137–140. [Online]. Available: http://dx.doi.org/10.1145/944592.944600

[154] P. Sharma, E. Perry, and R. Malpani, "IP multicast operational network management: design, challenges, and experiences," *Network, IEEE*, vol. 17, no. 2, pp. 49 – 55, mar/apr 2003.

[155] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," Internet Engineering Task Force, RFC 2212, Sept. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2212.txt

[156] S. Shenker and J. Wroclawski, "General Characterization Parameters for Integrated Service Network Elements," Internet Engineering Task Force, RFC 2215, Sept. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2215.txt

[157] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," in *Proceedings of IEEE INFOCOM*, March 2004.

[158] Sirpent, "Portable High-density Network Perf. Analysis System: SmartBits 600B," 2010. [Online]. Available: http://www.spirent.com/documents/1374.pdf

[159] A. Sobeih, W. Yurcik, and J. C. Hou, "VRing: A Case for Building Application-Layer Multicast Rings (Rather Than Trees)," in *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, Washington, DC, USA, 2004, pp. 437–446.

[160] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.

[161] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable Peer-to-Peer lookup Protocol for Internet Applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, February 2003. [Online]. Available: http://dx.doi.org/10.1145/638334.638336

[162] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "OverQoS: offering Internet QoS using overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 11–16, 2003. [Online]. Available: http://doi.acm.org/10.1145/774763.774764

[163] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: an efficient Peer-to-Peer Scheme for Media Streaming," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, 2003, pp. 1283–1292. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1208964

[164] "Universal TUN/TAP driver," 2010. [Online]. Available: http://vtun.sourceforge.net/tun/

[165] "UPnP Device Architecture 1.0, UPnP Forum," 2010. [Online]. Available: http://www.upnp.org

[166] A. Varga, "The OMNET++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference*. Prague, Czech Republic: SCS – European Publishing House, June 2001, pp. 319–324.

[167] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium: ICST (Institute for

Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.

[168] "VAST scalable Peer-to-Peer (P2P) Network for Virtual Environments (Virtual Worlds, MMOG and Simulations)," 2010. [Online]. Available: http://vast.sourceforge.net/

[169] "LBNL Audio Conferencing Tool (vat)," 2010. [Online]. Available: http://ee.lbl.gov/vat/

[170] "vat visual audio tool," 2010. [Online]. Available: ftp://ftp.ee.lbl.gov/conferencing/vat

[171] "UCB/LBNL Video Conferencing Tool (vic)," 2010. [Online]. Available: http://cobweb.ecn.purdue.edu/~ace/mbone/mbone/vic/vic.html

[172] "vic (video conferencing tool)," 2010. [Online]. Available: ftp://ftp.ee.lbl.gov/conferencing/vic

[173] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock, "Dr. Multicast: Rx for data center communication scalability," in *LADIS '08: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. New York, NY, USA: ACM, 2008, pp. 1–12.

[174] "Video Lan Client," 2010. [Online]. Available: http://www.videolan.org

[175] G. Wagenknecht, M. Anwander, M. Brogle, and T. Braun, "Reliable Multicast in Wireless Sensor Networks," in *7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*. Freie Universität Berlin, Fachbereich Mathematik und Informatik, September 2008, Tech. Report B 08-12.

[176] R. Wattenhofer, "3GP2P: 3rd Generation Peer-to-Peer Systems," 2010. [Online]. Available: https://www.rdb.ethz.ch/projects/project.php?proj_id=17835

[177] B. Waxman, "Routing of multipoint connections," *Broadband switching: architectures, protocols, design, and analysis*, pp. 347–352, 1991.

[178] "wb white board," 2010. [Online]. Available: ftp://ftp.ee.lbl.gov/conferencing/wb

[179] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," Internet Engineering Task Force, RFC 2211, Sept. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2211.txt

[180] C.-J. Wu, C.-Y. Li, and J.-M. Ho, "Improving the Download Time of BitTorrent-Like Systems," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 1125–1129. [Online]. Available: http://dx.doi.org/10.1109/ICC.2007.191

[181] L. Xiao, Y. Liu, W. Gu, D. Xuan, and X. Liu, "Mutual anonymous Overlay Multicast," *J. Parallel Distrib. Comput.*, vol. 66, no. 9, pp. 1205–1216, 2006.

[182] S. Yan, M. Faloutsos, and A. Banerjea, "QoS-aware Multicast routing for the Internet: the design and evaluation of QoSMIC," *IEEE/ACM Trans. Netw.*, vol. 10, no. 1, pp. 54–66, 2002. [Online]. Available: http://dx.doi.org/10.1109/90.986530

[183] J. Zhang, L. Liu, C. Pu, and M. Ammar, "Reliable Peer-to-Peer End System Multicasting through Replication," in *Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, Washington, DC, 2004, pp. 235–242.

[184] R. Zhang and Y. C. Hu, "Borg: a hybrid Protocol for scalable Application-Level Multicast in Peer-to-Peer Networks," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, New York, NY, USA, 2003, pp. 172–179. [Online]. Available: http://doi.acm.org/10.1145/776322.776349

[185] X. Y. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu, "A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance," *IEEE journal on selected areas in communications*, vol. 22, no. 1, pp. 18 – 28, January 2004.

[186] X. Zhang, X. Li, and Q. Zhao, "Service and Management for Multicast Based Audio/Video Collaboration System on CERNET," in *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part IV*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 853–856.

[187] X. Zhang, D. Liu, and X. Li, "Design and Implement Controllable Multicast Based Audio/Video Collaboration," in *ICCS '07: Proceedings of the 7th international conference on Computational Science, Part IV*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 701–704.

[188] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep., 2001.

[189] Y. Zhao, Y. An, C. Wang, and Y. Gao, "A QoS-Satisfied Interdomain Overlay Multicast Algorithm for Live Media Service Grid," in *Grid and Cooperative Computing - GCC 2005*, 2005, pp. 13–24.

[190] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, 2001, pp. 11–20. [Online]. Available: http://dx.doi.org/10.1145/378344.378347

List of Publications

Referred Papers (Journals, Conferences, Workshops)

Marc Brogle, Alican Geycasar, Torsten Braun: MCFTP – Implementation and Evaluation of a Multicast File Transfer Protocol, to be submitted for publication

Marc Brogle, Dominic Paprtitz, Dragan Milic, Torsten Braun: MCFTP - A Multicast File Transfer Protocol for Efficient Data Dissemination, to be submitted for publication

Thomas Staub, Markus Anwander, Kurt Baumann, Torsten Braun, Marc Brogle, Kirsten Dolfus, Christian Félix, Paul Kim Goode: Connecting Remote Sites to the Wired Backbone by Wireless Mesh Access Networks, *16th European Wireless Conference, Lucca, Italy , April 12 - 15, 2010*

Marc Brogle, Andreas Rüttimann, Torsten Braun: Quality of Service for Overlay Multicast in Chord, *The Fifteenth IEEE Symposium on Computers and Communications (ISCC'10)*, Riccione, Italy, June 22 - 25, 2010

Marc Brogle, Sebastian Barthlomé, Torsten Braun: Quality of Service for Multicasting using NICE, *25th Symposium On Applied Computing (ACM SAC 2010)*, Sierre, Switzerland, March 22 - 25, 2010, ACM

Thomas Staub, Markus Anwander, Kurt Baumann, Torsten Braun, Marc Brogle, Pascal Dornier, Christian Félix and Paul Kim Goode: Wireless Mesh Networks - Connecting Remote Sites, *SWITCH Journal, Zurich, Switzerland, March, 2010, pp. 10–12*

Marc Brogle, Luca Bettosini, Torsten Braun: Quality of Service for Multicasting in Content Addressable Networks, *12th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS 09)*, Telecom Italia Future Centre, Venice, Italy, October 26 - 27, 2009, pp. 170–175, Springer LNCS 5842, ISBN 978-3-642-04993-4

Marc Brogle, Luca Bettosini, Dragan Milic, Torsten Braun: A Performance Comparison of Native IP Multicast and IP Multicast Tunneled through a Peer-to-Peer Overlay Network, *2009 International Workshop on Peer-To-Peer Networking (P2PNet'09) at International Conference on Ultra Modern Telecommunications (ICUMT) 2009*, St. Petersburg, Russia, October 12 - 14, 2009, ISBN 9781-4244-3941-6

Marc Brogle, Dragan Milic, Torsten Braun: Multicast-Middleware der Universität Bern, Schweiz, resultierend aus EU FP6 IST IP "EuQoS", *PIK - Praxis der Informationsverarbeitung und Kommunikation*, Vol. 32, Nr. 3, October 5, 2009, pp. 176–178, Walter de Gruyter, ISSN 0930-5157 Gewinner des "KuVS Communication Software Preis" 2009

Thomas Staub, Marc Brogle, Kurt Baumann, Torsten Braun: Wireless Mesh Networks for Interconnection of Remote Sites to Fixed Broadband Networks, *Third ERCIM Workshop on eMobility*, University of Twente, Enschede, The Netherlands, May 27 - 28, 2009, pp. 97–98, ISBN 978-90-365-2846-7

Marc Brogle, Dragan Milic, Torsten Braun: Quality of Service for Peer-to-Peer based Networked Virtual Environments, *P2P-NVE 2008 Workshop at the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS 08)*, Melbourne, Victoria, Australia, December 8 - 10, 2008, pp. 847–852, IEEE, ISBN 978-0-7695-3434-3

Gerald Wagenknecht, Markus Anwander, Marc Brogle, Torsten Braun: Reliable Multicast in Wireless Sensor Networks, *7. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, Berlin, Germany, September 25 - 26, 2008, pp. 69–72, Freie Universität Berlin, Fachbereich Mathematik und Informatik, Tech. Report B 08-12

Torsten Braun, Marc Brogle, Patrick Lauer: Peer-to-Peer-Netze: Informationen effizient im Internet verbreiten, *Bulletin SEV/VSE*, Vol. 07, Nr. 21, December, 2007, pp. 9–12, Electrosuisse, ISSN 1660-6738

Marc Brogle, Dragan Milic, Torsten Braun: Supporting IP Multicast Streaming Using Overlay Networks, *QShine: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Vancouver, British Columbia, Canada, August 14 - 17, 2007, ICST, ISBN 978-1-59593-756-8

Marc Brogle, Dragan Milic, Torsten Braun: QoS Enabled Multicast for Structured P2P Networks, *P2PM Workshop at the 4th IEEE Consumer Communications and Networking Conference*, Las Vegas, NV, USA, January 11 - 13, 2007, pp. 991–995, IEEE, ISBN 1-4244-0667-6

Dragan Milic, Marc Brogle, Torsten Braun: Video Broadcasting using Overlay Multicast, *Seventh IEEE International Symposium on Multimedia (ISM 2005)*, Irvine, California, USA, December 12 - 14, 2005, pp. 515–522, IEEE Computer Society Press, ISBN 0-7695-2489-3

Manuel Günter, Marc Brogle, Torsten Braun: Secure Communication: A new Application for Active Networks, *International Conference on Networking (ICN'01)*, Colmar, France, July 9 - 13, 2001, pp. 206–216, Springer LNCS 2094, ISBN 3-540-42303-6

Editorial Activities (Workshop Proceedings, Technical Reports)

Marc Brogle, Torsten Braun: *BeNeFri Summer School 2009 on Dependable Systems*, Münchenwiler, Switzerland, September 8, 2009, IAM-09-006

Marc Brogle, Geert Heijenk, Torsten Braun, Dimitri Konstantas: *Proceedings of the Third ERCIM Workhop on eMobility*, University of Twente, Enschede, The Netherlands, May 27 - 28, 2009, ISBN 978-90-365-2846-7

Marc Brogle, Dragan Milic, Markus Anwander, Gerald Wagenknecht, Markus Waelchli, Torsten Braun, R. Kummer, Markus Wulff, R. Standtke, H. Sturzrehm, E. Riviere, P. Felber, S. Krenn, C. Ehret, C, Latze, Philipp Hurni, Thomas Staub: *BeNeFri Summer School 2008 on Dependable Systems*, Quarten, Switzerland, November 18, 2008, IAM-08-003

Torsten Braun, Ulrich Ultes-Nitsche, Marc Brogle, Dragan Milic, Patrick Lauer, Thomas Staub, Gerald Wagenknecht, Markus Anwander, Markus Waelchli, Markus Wulff, Carolin Latze, Michael Hayoz, Christoph Ehret, Thierry Nicola: *RVS Retreat 2007 at Quarten*, Quarten, Switzerland, December, 2007, IAM-07-004

Unreferred Papers (Technical Reports, Project Deliverables)

Marc Brogle: Quality of Service for "NICE" Overlay Multicasting, BeNeFri Summer School 2009 on Dependable Systems, Münchenwiler, Switzerland, September 8, 2009, IAM-09-006

Marc Brogle: OM-QoS: Overlay Multicast Quality of Service, BeNeFri Summer School 2008 on Dependable Systems, Quarten, Switzerland, November 18, 2008, IAM-08-003

Wojciech Burakowski, Jordi Mongay Batalla, Marc Brogle, et al: Report on scalability evaluation of EuQoS system, EuQoS Report, January 25, 2008

José Enríquez, María Ángeles Callejo, Marc Brogle, Dragan Milic, et al: Annex to D1.2.2: EuQoS Architecture update for , Annex to EuQoS Deliverable D1.2.2, CEC Deliverable Number 004503/TID/DS/D1.2.2/A2 - ANNEX, December 28, 2007

José Enríquez, María Ángeles Callejo, Marc Brogle, Dragan Milic, et al: EuQoS Architecture update for Phase 2, EuQoS Deliverable D1.2.2, CEC Deliverable Number 004503/TID/DS/D1.2.2/A2, December 28, 2007

Donal Morris, Marc Brogle, Dragan Milic, et al: Annex to D1.2.3: Exploitation Cookbook, Final, EuQoS Deliverable D1.2.3, CEC Deliverable Number 004503/ TID/DS/D1.2.3/A1 - ANNEX, December 28, 2007

Halina Tarasiuk and Wojciech Burakowski, Marc Brogle, Dragan Milic, et al: Methodology for testing EuQoS system, EuQoS Deliverable D2.2.5, CEC Deliverable Number 004503/WUT/DS/D2.2.5./A2, December 28, 2007

María Ángeles Callejo, José Enríquez, Marc Brogle, Dragan Milic, et al: Annex 1 to D3.2.5: Implementation Final Report - Detailed design, EuQoS Deliverable D3.2.5, CEC Deliverable Number 004503/ED/DS/D3.2.5/A1 - Annex 1, December 28, 2007

María Ángeles Callejo, José Enríquez, Marc Brogle, Dragan Milic, et al: Annex 2 to D3.2.5: Implementation Final Report - EuQoS user's manual, EuQoS Deliverable D3.2.5, CEC Deliverable Number 004503/ED/DS/D3.2.5/A1 - Annex 2, December 28, 2007

Olivier Dugeon, Marc Brogle, Dragan Milic, et al: Prototype P#4 tests report , EuQoS Deliverable D5.2.3, CEC Deliverable Number 004503/FTRD/DS/D5.2.3/A1, December 28, 2007

Olivier Dugeon, Marc Brogle, et al: EuQoS System Demonstrations Report for Phase II, EuQoS Deliverable D5.2.4, CEC Deliverable Number 004503/FTRD/DS/D5.2.4/A1, December 28, 2007

Thomas Staub, Marc Brogle, et al: Report on teaching experiences of the e-learning course, the improvements to be done and the improvements achieved, EuQoS Deliverable D6.2.4, CEC Deliverable Number 004503/UBern/DS/D6.2.4/A1, December 28, 2007

Marc Brogle: OM-QoS: Overlay Multicast Quality of Service, RVS Retreat 2007 at Quarten, December, 2007, IAM-07-004

Pascal Le Guern, Olivier Dugeon, Marc Brogle, Dragan Milic, et. al.: Trial report release 2, EuQoS Deliverable D5.1.5, CEC Deliverable Number 004503/FTRD/DS/D5.1.5/A1, February 15, 2007

Donal Morris, Thomas Staub, Marc Brogle, et. al.: Second summary of standardization documents, EuQoS Deliverable D6.2.2, CEC Deliverable Number 004503/REDZINC/DS/D6.2.2/A1, January 31, 2007

José Enríquez, María Ángeles Callejo, Marc Brogle, Dragan Milic, et. al.: EuQoS Architecture update for Phase 2, EuQoS Deliverable D1.2.2, CEC Deliverable Number 004503/TID/DS/D1.2.2/A1, January 31, 2007

José Enríquez, María Ángeles Callejo, Marc Brogle, Dragan Milic, et. al.: Annex to D1.2.2: EuQoS Architecture update for Phase 2, EuQoS Deliverable D1.2.2, CEC Deliverable Number 004503/TID/DS/D1.2.2/A1 - ANNEX, January 31, 2007

Thomas Staub, Marc Brogle, et. al.: Report on teaching experiences of the e-learning course, the improvements to be done and the improvements achieved as well as the newly produced e-learning modules, EuQoS Deliverable D6.2.1, CEC Deliverable Number 004503/UoB/DS/D6.2.1/A1, December 26, 2006

Pascal Le Guern, Olivier Dugeon, Marc Brogle, Dragan Milic, et. al.: Testbed integration test plan Release 2, EuQoS Deliverable D5.1.4, CEC Deliverable Number 004503/FTRD/DS/D5.1.4/A2, December 4, 2006

Mark Günther, Martin Potts, Marc Brogle, Thomas Staub, et al.: Report on Raising Public Participation and Awareness, EuQoS Deliverable D6.1.6, CEC Deliverable Number 004503/Telscom/DS/D6.1.6/A1, April 30, 2006

Pascal Le Guern, Marc Brogle, Dragan Milic, et al.: Trial Report, EuQoS Deliverable D5.1.5, CEC Deliverable Number 004503/FTRD/DS/D5.1.5/A1, April 26, 2006

Pascal Le Guern, Marc Brogle, Dragan Milic, et. al.: Testbed integration test plan, EuQoS Deliverable D5.1.4, CEC Deliverable Number 004503/FTRD/DS/D5.1.4/A1, April 13, 2006

Pascal Le Guern, Marc Brogle, Dragan Milic, et al.: Connectivity and Performance Tests Report for Local and Pan-European (across GEANT) Testbed Design for the Trial, EuQoS Deliverable D5.1.2, CEC Deliverable Number 004503/FTRD/DS/D5.1.2/A3, February 6, 2006

Pascal Le Guern, Marc Brogle, Dragan Milic, et. al.: First individual based EuQoS System test report, EuQoS Deliverable D5.1.3, CEC Deliverable Number 004503/FTRD/DS/D5.1.3/A2, February 6, 2006

Laurent Baresse, Enrico Angori, Giuseppe Martufi, Marc Brogle, Dragan Milic, et. al.: Extended QoS API and Middleware layer for phase 1 application use-cases, EuQoS Deliverable D3.1.1, CEC Deliverable Number 004503/Partner/DS/D3.1.1/A2, August 1, 2005

José Enríquez, Marc Brogle, Dragan Milic, et. al.: Business models and system design specification, EuQoS Deliverable D1.1.3, CEC Deliverable Number 004503/TID/DS/D1.1.3/A2, August 1, 2005

Marc Brogle, Dragan Milic: MC-FTP: A Multicast File Transfer Protocol for Efficient Data Dissemination, RVS Retreat 2005 at Griesalp, June 27 - 30, 2005, IAM-05-002

Enrico Angori, Giuseppe Martufi, Marc Brogle, Dragan Milic, et. al.: System Design: Functions, Interfaces Specification, EuQoS Deliverable D1.1.2, CEC Deliverable Number 004503/TID/DS/D1.1.2/A2, May 13, 2005

Régis Frechin, Pascal Le Guern, Marc Brogle, Dragan Milic, et al.: Technical requirements for the trial, tasks and scheduling, EuQoS Deliverable D5.1.1, 004503/FTRD/DS/D5.1.1/A1, March 1, 2005

Pascal Le Guern, Marc Brogle, Dragan Milic, et al.: First Individual Based EuQoS System Test Report, EuQoS Deliverable D5.1.2, CEC Deliverable Number 004503/FTRD/DS/D5.1.2/A2, March 1, 2005

José Enríquez, Jorge Andrés, Marc Brogle, Dragan Milic, et. al.: Definition of Business, Communication and QoS models - Intermediate, EuQoS Deliverable D1.1.1, CEC Deliverable Number 004503/TID/DS/D1.1.1/A2, March 1, 2005

José Enríquez, Jorge Andrés, Marc Brogle, Dragan Milic, et. al.: Annex to D1.1.1: Definition of Business, Communication and QoS models - Intermediate, EuQoS Deliverable D1.1.1, CEC Deliverable Number 004503/TID/DS/D1.1.1/A1, March 1, 2005

Manuel Günter, Marc Brogle, Torsten Braun: Secure Communication: a New Application for Active Networks, July, 2000, IAM-00-007

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: Brogle Marc

Matrikelnummer: 95-101-085

Studiengang: Informatik

Bachelor ☐ Master ☐ Dissertation ☒

Titel der Arbeit: IP Multicast using Quality of Service enabled Overlay Networks

LeiterIn der Arbeit: Prof. Dr. Torsten Braun

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 29.04.2010

Ort/Datum

Unterschrift

Curriculum Vitae

Personal Details

Name	Marc Brogle
Date of Birth	December 25, 1973
Address	Centralbahnplatz 12
	CH-4051 Basel, Switzerland
Hometown	Sisseln AG, Switzerland
Nationality	Swiss

Education

2004 – 2010	Ph.D. student in Computer Science at the University of Bern, Switzerland
2004	Master of Science in Computer Science, University of Bern, Switzerland
1995 – 2001	Study of Computer Science at the University of Bern, minor fields in Mathematics and Microelectronics
1988 – 1994	Grammar school, Biel