Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Communications Systems Research Group

Thesis to the Semester Project

Towards Resilience Against Node Failures in Overlay Multicast Schemes

Simon Heimlicher

Supervision: Kostas Katrinis, Prof. Dr. Bernhard Plattner

Publication: 11th August 2004

Contents

List of Figures

Abstract

Several emerging Internet applications require group communication. *Multicast*, a many-to-many communication service model for the *Internet Protocol (IP)*, was proposed fifteen years ago, but still lacks universal deployment. Consequently, researchers have investigated other solutions, *overlay multicast* being the most prominent among them. In Internet terminology, an *overlay* is a virtual network built on the IP substrate. For multicast in particular, it is usually a *spanning tree* formed by the hosts belonging to the *multicast group* and the IP unicast links among them.

In the present semester project, a new scheme based on *Narada [1]* has been designed and evaluated using simulations. The primary design goal was resilience against member failures. Since packet forwarding is performed by ordinary group members, member failure leads to loss of data and consequently to quality degradation for the target application. The novel features of the present approach compared to Narada are overlay optimization based on negotiation among neighbors, triggered routing updates for fast convergence and forced route changes through poisoning.

Simulation experiments have shown that in case of abrupt node failure, packet loss is unacceptably high for typical multicast applications. With forwarding entities as unreliable as personal computers, *IP-level multi-path* routing, which allows for graceful degradation on single path failure, appears to be a valuable solution.

Preface

This work started in October 2003 and culminated with the thesis at hand in July 2004. The original specification of the subject was:

> *Overlay Multicast Simulation:* Problems in the deployment of network layer multicast have led the research community to alternative group communication solutions. One of them is application layer multicast, where data is routed via an application layer overlay to the group members. The goal of this thesis is to model and implement a generic simulation framework for such overlays.

This problem was then refined in the conceptual formulation[1] with the title *"Evaluation of Routing Schemes for Group Communication in Packet Switched Networks."* The formulation of the problem to be addressed was:

> *What is the most efficient multicast service for wide deployment in the existing Internet* [2] *among the existing alternatives?*

The procedure to achieve this goal was planned to follow these steps: 1. Studying of related work. 2. Selection of a representative set of multicast schemes to put under study. 3. Discussion of the evaluation parameters and creation of application profiles. 4. Studying the use of the OMNeT^{++} simulator. 5. Implementation of the chosen protocols in the OMNeT^{++} environment and measurement of the variables of interest. 6. Processing of the results and derivation of conclusions.

After the first three steps, we were planning to compare two overlay multicast schemes with IPv4 multicast. However, in the process of implementing the first overlay multicast scheme, we noticed that it was infeasible to implement the other two in the limited time frame of a semester project. Thus, we decided to focus on the implementation of the first overlay multicast protocol and then compare it with an existing implementation of the IPv4 multicast scheme. In the process of the implementation, we came up with a few improvements over the original design. The simulation experiments with our scheme yielded very interesting results and we are currently in the process of putting the outcome of our work into a manuscript that we plan to submit for publication in the near future.

It is our perception that the audience of semester theses is usually a very limited set of people: the supervisor and maybe some fellow students or the parents of the author. Most theses assume a decent knowledge in the subject area and this makes them a frustrating read for people from other fields. Having this in mind, we tried to keep our introduction to network protocols and overlay multicast schemes comprehensible to the broad audience. Readers with diverse background are encouraged to start their journey through this text with Chapter 2 on page 3 and then continue with the introduction on the next page. This approach should make the remaining chapters more enjoyable.

Welcome to the world of overlay multicast!

Simon Heimlicher and Kostas Katrinis
11th August 2004.

[1] The complete document can be found in Appendix A.

[2] We use the term *Internet* with a capital 'I' when we talk about the global internetwork commonly referred to as "the Internet" that evolved from *ARPAnet*, developed by the *Advanced Research Projects Agency (ARPA)* in the USA in the 1960s[2].

Chapter 1

Introduction

The growth rate of the Internet during the last decade has been spectacular. One of the key characteristics that enabled the transformation from an internetwork used almost exclusively by military and educational institutes to the omnipresent global Internet of today is the simplicity of its underlying protocol suite, TCP/IP. At its core lies the *Internet Protocol (IP)*, which does very little, but does it extremely well: it delivers short messages from a source machine A across all intermediate networks to a destination machine B.

As the Internet evolved, so did the expectations of its users and the demands of the services running over it. Emerging applications like audio/video streaming and conferencing, distributed computing and database replication require data-intensive communication among large and heterogeneous groups of hosts.

With the advent of broadband Internet access to the homes of more and more people, the audience for such data-intensive applications grows rapidly. The basic approach of replicating data packets at the source and delivering every single copy separately is very expensive in terms of network resources. Since the connections to the end systems are getting faster at a similar pace as the servers and backbone networks, at some point in time, this will no longer be feasible using only unicast communication.

Clearly, a more intelligent distribution scheme is needed.

Multicast is a service model for the distribution of data from one source to a group of end systems. Even though it has been proposed fifteen years ago and today almost every router has built-in support for multicast forwarding, providers of rich content and value-added services still do not rely on it due to incomplete deployment.

In this chapter, we will give an overview of the currently available network layer multicast protocols and their limitations and show where overlay multicast schemes come into play.

For readers not familiar with networking protocols and multicast, it is suggested to first read Chapter 2 on page 3.

1.1 Motivation

The core network protocol of today's Internet, *IPv4 [3]*, offers native support for a many-to-many service model termed *multicast*. Unfortunately, this extension to the TCP/IP protocol suite dating back to 1988 has never been adopted by its target audience, the Internet service providers. To make matters worse, several IPv4 multicast protocols exist currently. The most popular are in chronological order of occurence: *Distance Vector Multicast Routing Protocol (DVMRP) [4], Multicast Open Shortest Path First (MOSPF) [5], Core Based Trees (CBT) [6], [7], [8]* and *Protocol Independent Multicast (PIM) [9], [10]*. The next version of the IP protocol, *IPv6* [11], is currently being deployed Internet-wide. This protocol version will provide much more sophisticated multicast services from the beginning. While IPv4 multicast limits the maximal number of simultaneously active groups to about 2^{20}, the address range IPv6

reserves for multicast (about 2^{112} addresses) should be large enough for the next few decades.

Researchers have investigated other approaches in the meantime. Since increasingly deployed peer-to-peer file sharing protocols like Gnutella [12] and BitTorrent [13] are already quite efficient even though they run in the application layer on end systems, it appears feasible to also implement multicast services in this manner. At the time of writing, however, no protocol suite for application layer multicast has been adopted by the IETF [14] or any other standards committee.

A considerable amount of work has been done by researchers to analyze different approaches to the multicast problem, but a lot of questions remain unanswered. The question addressed by this thesis is:

What is the **most efficient** multicast service
for wide deployment in the existing Internet among the existing alternatives?

Our approach towards answering this question was in brief: Based on a set of requirement profiles of typical multicast applications, we rated qualitatively most of the currently publicly available application layer multicast protocols. We decided to use *Narada* [1], also known as *End System Multicast (ESM)* as the basis for our simulation. We implemented Narada in the *OMNeT*$^{++}$ [15] discrete event system simulator. For the generation of the Internet-like simulation topologies, we used *GT-ITM* [16]. We simulated a multicast application in an Internet-like topology. Finally, we weighted the results of the simulation experiments according to our profiles and drew conclusions about the suitability of our approach for a number of multicast applications.

The present thesis is structured as follows. In Chapter 2, we will present the basic concepts of network protocols, IPv4 multicast and introduce overlay multicast. Chapter 3 provides an overview of the multicast schemes we have considered and descriptions of the protocols we have put under study. Subsequently, in Chapter 4, we specify our evaluation methodology and the simulated network topology. We discuss the results of the simulation experiments in Chapter 5 and assess the performance of our scheme for various applications. We conclude in Chapter 6.

Appendix A contains the complete initial conceptual formulation. In Appendix B, we review the goals of the project we reached and those we haven't achieved. Appendix C provides an explanation of the parameters of our OMCP implementation and excerpts from the C^{++} source code used for the simulation in the OMNeT^{++} environment.

Chapter 2

Background Information

This chapter introduces the fundamental terms and concepts we are going to touch in the thesis. We first give a very brief overview of network protocols and the concept of multicast communication. In Section 2.2, we outline the history of multicast in the Internet. The motivation for our work is given in Section 2.3. We conclude the chapter with a primer on the core subject—Overlay Multicast—in Section 2.4.

2.1 Introduction to Network Protocols

A thorough introduction to computer networks is beyond the scope of this text. Nevertheless, we will try to explain the essential characteristics of network protocols in this section. A broad overview of the technologies used at the various network layers is given in [17]. A system-oriented discussion of the important concepts of computer networks can be found in [18]. To keep this section brief enough, we will take the liberty of skipping concepts that are not of prime importance to the context. The footnotes give additional information where we omit important details.

This section is organized as follows. First, we give an overview of the concepts of computer networks, then we explain the terms relevant to the thesis in more detail.

2.1.1 Definitions

First of all, we need to give some definitions of basic components of networks we will refer to in the future.

Network The general term *network* refers to any means which allows two or more computers to communicate with each other.

Protocol When people communicate with each other, they use a language which allows them to process the acoustic waves received by their ears or the symbols seen by their eyes. To understand each other, computers need to use a common language, too. Since we assume that computers don't have any intuition in the sense that they are unable to read between the lines, computers can only communicate using a very strict kind of language. Such languages are called *protocols.*

When discussing computer networks, a distinction between the following classes of devices is often made based on their purpose.

Host A *host* or *end system* is either a personal computer or a server. On an abstract level, we may also call a host a *node.*

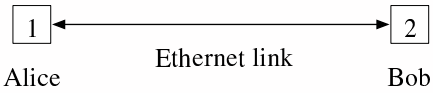


Figure 2.1: *An Ethernet link between hosts 1 and 2.*

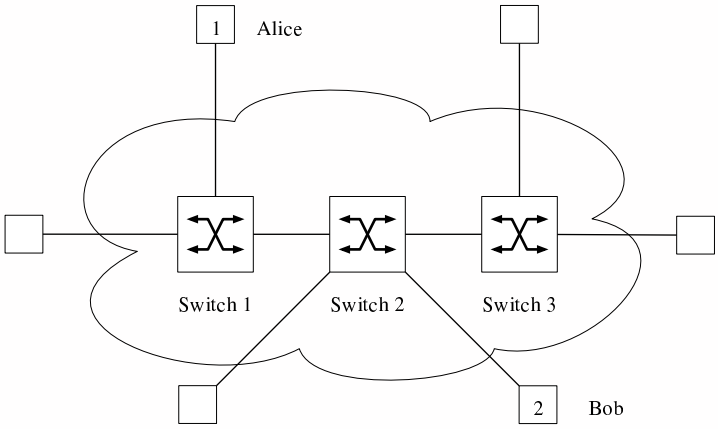


Figure 2.2: *A simple network connected by three switches.*

Link The basic building blocks for networks are *links.* A link is an abstract word for the physical medium that carries the signals between devices. This may be a cable or a wireless connection. The most common medium is copper wire. *Ethernet* is the most popular link techonology for home and office networks. Ethernet uses copper wire or glass fibre or, in the wireless case, no medium at all. To simplify the discussion, we will assume wired networks in this section. In Figure 2.1, a simple link between the computers of Alice and Bob is shown.

Switch Switches are active hardware devices which connect hosts to each other. The resulting network is called a *subnetwork.* Figure 2.2 is an example of how several hosts may be connected to each other using switches. The delivery of data is performed by the switches entirely transparent to the end systems and without a perceiveble delay. [1] In contrast to routers, which will be discussed next, a switch can only handle messages to hosts which are directly connected to it or another directly connected switch, i.e., hosts in the same network.

The cloud symbol in Figure 2.2 denotes any kind of network and is commonly used to depict the Internet. It just interconnects all systems which have a connection to it.

Router *Routers* are active hardware devices which interconnect two or more independent networks. A network of networks is termed an *internetwork.* Routers are able to connect networks of different kind.

[1] A *hub* is a passive network device which has the same purpose as a switch. The difference between a hub and a switch is, that a hub only supports communication between two ports at a time. All data which are received on one port are sent out on all other ports. In contrast, a switch with n ports supports up to $\frac{n}{2}$ concurrent connections at full speed in both directions. When a switch first receives a message from a host with the physical address A, addressed at the host with physical address B, it sends this message on all interfaces except the one where the message came from (exactly like a hub). The switch takes note of the destination port and the corresponding physical address. In the future, when the switch receives a message to an address it has already seen, it transmits it only to the corresponding port.

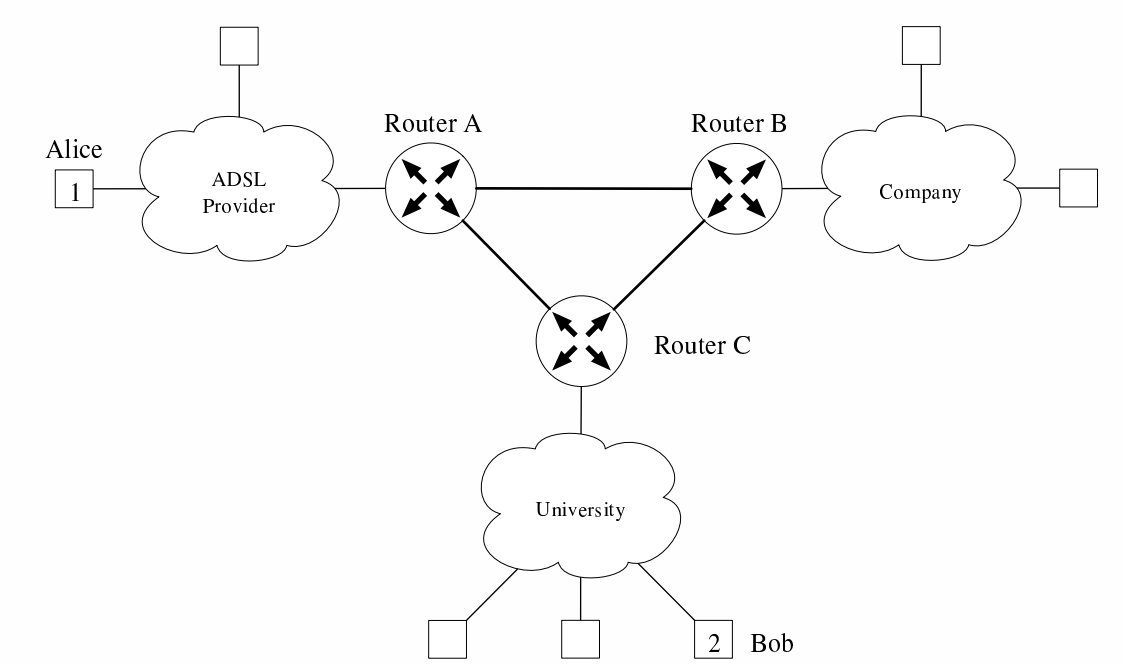


Figure 2.3: *A simple internetwork comprised of the three networks "ADSL Provider", "Company" and "University" and connected by three routers. Thus, Alice and Bob can communicate with each other, even though they are connected to different networks.*

Therefore, they are usually a lot more sophisticated than switches. A very simple internetwork is shown in Figure 2.3. Note that this internetwork contains a loop. Routers are able to cope with this situation. Communication between Alice and Bob normally runs over router A. But if the physical link between router A and router C fails, the routers will adapt to the situation quickly and future messages will be sent via router B and router C.

When an application, e.g. an e-mail client, sends a message to the e-mail server running on another host, the message passes the following facilities before it is actually transmitted on the link:

Network Protocol Stack This is usually a part of the operating system software. It knows how to communicate with other computers which have the same protocol stack built in. In the Internet, the *TCP/IP [3], [19]* protocol stack is used.

Network Interface Driver The piece of software which translates the messages into commands that are understood by the network hardware is called network interface driver. It is situated below the device-independent network protocol stack and above the hardware responsible for network communication.

Network Interface This part of the computer hardware that transforms the data to be sent into electrical or electro-magnetical signals when transmitting and restores the data from these signals when receiving. Most network interfaces today are able to send and receive at the same time.

We will next discuss the concepts pertaining to the current thesis in more detail.

2.1.2 Switching

Historically, the process of preparing the path of a message through the network is called *switching.* Networks can roughly be divided into two classes by the switching strategy they use. Until the 1950s, most networks were *circuit-switched*. This means, that a physical path between sender and receiver is set up for and dedicated to the connection before two hosts communicate with each other—very similar to how telephone operators in the early days established the telephone connection on request of the caller. Once this has been accomplished, the sender may inject messages into the connection at will and no other entity can use the physical lines involved in this connection. After all data have been sent, the connection is torn down explicitly. No congestion can occur and all bytes of the message arrive in sequence. Circuit switching is used in telephone networks[2].

In contrast to the telephone network, current computer networks deliver data in the following way: The sender cuts the message into small pieces called *packets* and prepares them for transmission by prepending a header indicating the destination address to every piece. The network infrastructure then routes these packets along potentially disjoint paths through the network to the receiver, which reassembles them to the original message. This type of network is called *packet-switched [20],* because it operates on individual packets. With packet switching, there is no need to set up the path for a message beforehand. Instead, the first packet can be sent off as soon as it becomes available. But at no point in the transmission is it guaranteed that the network does have enough free resources to deliver the message. If too many messages are injected into the network, congestion occurs and packets need to be buffered or even dropped at the bottleneck. Usually this happens at a router whose input buffer is full.

Since packet-switched networks make far better use of the network resources available and allow for greater throughput, today's computer networks are usually packet-switched. In this text, we always assume packet-switched networks.

2.1.3 Protocol Stack

There are a lot of analogies between human and computer communication. In a simple transmission of a message from Alice to Bob, several distinct steps can be distinguished: At the beginning, Alice's brain translates her thought into words, for example, "Hello Bob". Then, it sends an electrical signal to the vocal cords which orders them to generate the sound of these words. This sound wave is carried to Bob by the physical medium, the air. His eardrums translate this mechanical wave back into an electrical signal, which can then be processed by his brain and the original thought Alice had in mind is restored. Note that the actual message is transformed from a thought into a mechanical wave in a few steps, transmitted over the air, and then transformed back in similar steps into a thought.

Similar transformations are applied to messages sent over a network by the *protocol stack*[3]. As the name implies, we can think of it as a stack of protocols. The stages which a message passes when it is transformed are called *layers* or *levels.*

[2]Today, however, even telephone networks are mostly packet-switched. It is possible to unite the advantages of both types with *virtual circuits:* A logical connection is set up between two end points and the necessary resources are allocated for the duration of the connection. *X.25* is a popular example for this kind of network. It supports permanent virtual circuits which are an alternative for leased lines. The *Asynchronous Transfer Mode (ATM)* network standard was developed to allow the multiplexing of thousands of telephone connections into one optical fibre link. It uses packet-switching to accomplish this and a major part of the Internet backbone consists of ATM links. In Switzerland, only the last mile from the phone jack to the first device of the telephone network is analog, the interconnection network is completely digital. *Digital Subscriber Line (DSL)* and its siblings *ADSL, HDSL* etc. make use of this fact.

[3]There exist several reference models for protocol stacks. The most popular model, at least for educational purposes, is the *OSI Reference Model [21], [22],* developed in 1983 by the International Organization for Standardization. OSI is an acronym for Open System Interconnection, and this model defines a networking framework for implementing services and protocols in seven layers. The Internet, however, uses the *TCP/IP* protocol stack. We will use a combination of both models because we think it is confusing to talk about two different models in an introductory text.

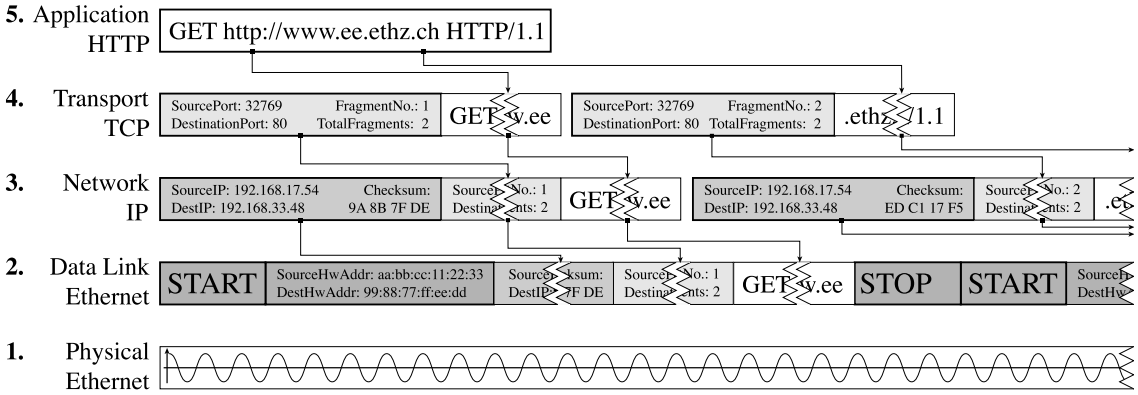


Figure 2.4: *A simplified example of a message traversing the TCP/IP protocol stack.*

A simplified example of what this looks like in the TCP/IP protocol stack is shown in Figure 2.4. Note that the TCP/IP reference model actually consists of four layers since it has only one layer below the network layer. The layer numbering shown in the figure, however, is the most common among network operators and network gear manufacturers and is often used in the literature, for example in the book by Tanenbaum [17].

Corresponding layers in different machines run the same protocol and communicate with each other. They cannot exchange messages directly, though. Instead, the message to send is passed to the lower layer which will further process it. It can thus be said that the lower layer provides a service to the upper layer. In the above example, the *application layer* cannot send the user's request for a web page to the application layer running on the web server directly. It can, however, instruct the *transport layer* to deliver the message to the transport level at the destination host. The transport layer itself does not actually transmit the message, though. Instead, it cuts the message into packets and hands these down to the *network layer,* which is responsible to do its best at delivering those packets to the destination. From the network layer, the message is again passed to the lower layer, the *data link layer,* which controls the access to the link, corrects bit errors, etc. It converts the message including all headers, error correction codes etc., into a sequence of bits, which is then translated into voltages by the *physical layer.* The physical medium—the *channel*—then carries the signal to the remote host.

Whenever a message is passed from an upper protocol layer to the one below, the upper layer first adds a *protocol header*[4] to the message. The lower layer then operates on the longer message including the header of the upper layer. Once the message has reached the lowest layer, it is transmitted to the destination machine. This machine then reverses the process: Every layer first reads the information in the header added by its peer at the sender, acts accordingly and strips the header off before passing the remainder of the message to the upper layer.

Whether a layer is implemented in physical or software depends on the device type. In an end system, e.g., on a personal computer or a web server, only the physical layer is implemented in physical. Data link, network and transport layers are provided by the operating system software. The application level protocol is running in the application program, e.g., the web browser or the e-mail client.

In the context of multicasting, we will mostly talk about the network layer protocol IP or application level protocols.

[4]Depending on the data link layer protocol, this layer may additionally append a trailer.

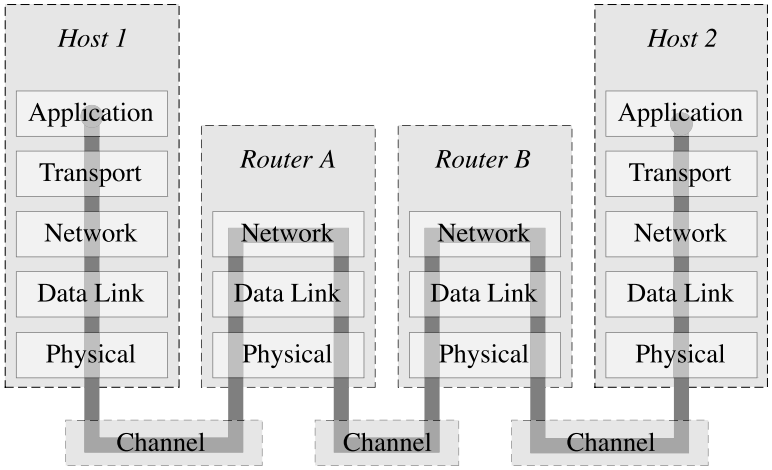


Figure 2.5: *A simplified example of the protocol stack traversals of a message from host 1 to host 2 via routers A and B.*

2.1.4 Routing and Forwarding

Routing is a joint venture of all routers in the network with the goal of distributing the information necessary to deliver packets to any destination host in the network. The path which a packet follows is called a *route.* The routing information is stored in the *routing table* of every router. *Forwarding* is a much simpler process. It just means to correctly guide packets through the network using the information in the routing table. To forward, only a subset of the data of the routing table is necessary and this subset is often called a *forwarding table* in this context. The forwarding table is established by the routing process or manually set up by the network operator.

In routers, the message is only passed from the physical to the data link and then to the network layer. In the network layer, the router examines the network layer destination address, looks up the *next hop* of the route and passes the message back to the data link layer with the instructions to send it to this host.

When a packet arrives, the router looks up the route to the destination in its routing table and sends the message on the correct interface into the next network. The machines along the route are also termed *hops.* Routing tables contain information about the *cost* of a route. The cost is an indication of how good a route is in the perception of the router. A very basic routing algorithm might just count the number of hops it takes to the destination and use this number as cost. When deciding which route it should store in the routing table, it would take the route with the least number of hops. This is called *shortest path* routing. In Figure 2.6, an example routing topology of a TCP/IP internetwork is shown. The cost of the links is shown in *italics* beside every link. The routing table of router A is printed below. The destination addresses refer to the networks. For instance, “10.1.0.0/16”, denotes the network with address 10.1.0.0 and a netmask of 16 bits. The latter means that the network part is defined by the first 16 bits. The rest of the address is the host address. The IP addressing scheme is further explained in the next section.

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.1.0.1	I01	0
10.2.0.0/16	10.3.0.1	I03	4
10.3.0.0/16	10.3.0.1	I03	1

If, however, the link between router A and router C fails, the routing tables have to adopt to the new situation. The modified routing tables are also shown in the figure. Routes that have changed are shown in *italics* in the modified routing table of router A:

Routing Table of Router A

All links up

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.1.0.1	I01	0
10.2.0.0/16	10.3.0.1	I03	4
10.3.0.0/16	10.3.0.1	I03	1

Link Router A – Router C down

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.1.0.1	I01	0
10.2.0.0/16	**10.2.0.1**	**I02**	**5**
10.3.0.0/16	**10.2.0.1**	**I02**	**8**

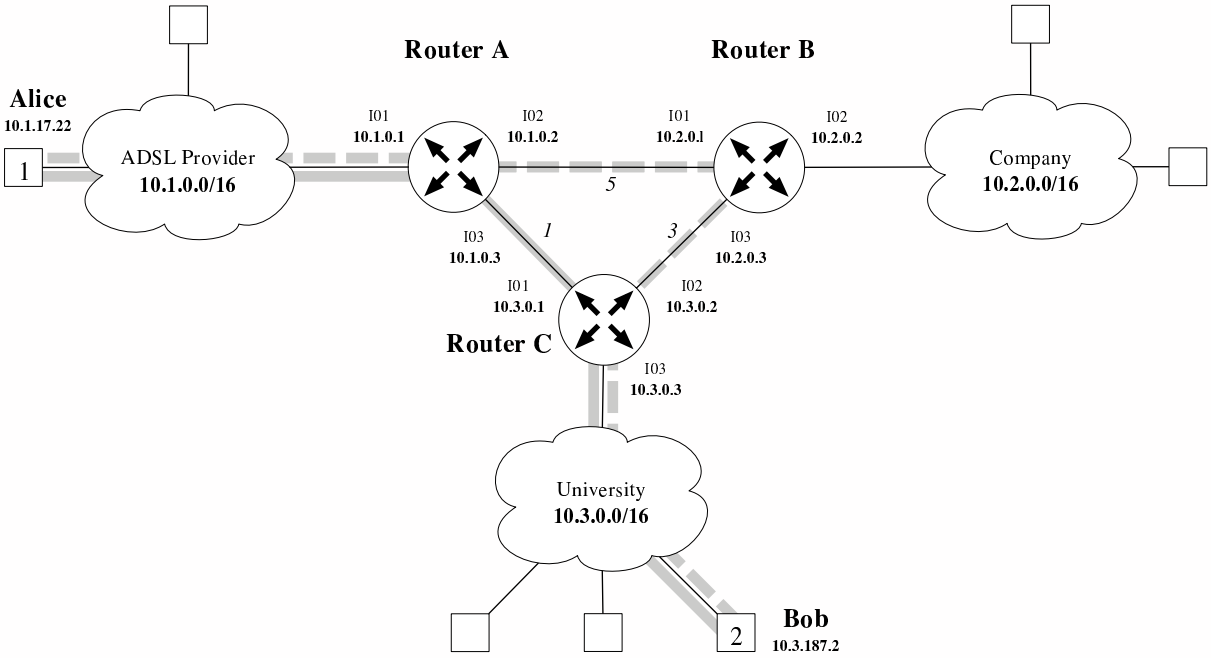
Routing Table of Router B

All links up

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.3.0.2	I03	4
10.2.0.0/16	10.2.0.2	I02	0
10.3.0.0/16	10.3.0.2	I03	3

Link Router A – Router C down

Destination	Next Hop	Interface	Cost
10.1.0.0/16	**10.1.0.2**	**I01**	**5**
10.2.0.0/16	10.2.0.2	I02	0
10.3.0.0/16	10.3.0.2	I03	3



Routing Table of Router C

All links up

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.1.0.3	I01	1
10.2.0.0/16	10.2.0.3	I02	3
10.3.0.0/16	10.3.0.3	I03	0

Link Router A – Router C down

Destination	Next Hop	Interface	Cost
10.1.0.0/16	**10.2.0.3**	**I02**	**8**
10.2.0.0/16	10.2.0.3	I02	3
10.3.0.0/16	10.3.0.3	I03	0

Figure 2.6: *A simplified example of routing. When all links are up, messages between Alice and Bob are sent via Router A and Router C. When a link failure between router A and C occurs, the routing tables change accordingly and traffic is forced to take the long route via router B. The modified routes are shown in bold type.*

Destination	Next Hop	Interface	Cost
10.1.0.0/16	10.1.0.1	I01	0
10.2.0.0/16	*10.2.0.1*	*I02*	*5*
10.3.0.0/16	*10.2.0.1*	*I02*	*8*

Note, that even though on paper this routing table update looks very straightforward, it is a big challenge in practice. There are many intricacies hidden in distributed algorithms. How some of them can be tackled with is shown in Section 3.3.

2.1.5 TCP/IP Protocol Suite

The protocol stack run by all computers connected to the Internet is called *TCP/IP* [3], [19] TCP/IP denotes a whole protocol family. To give an overview, we follow a message traversing the TCP/IP stack and refer again to Figure 2.4.

In the application layer, we see the actual message: The sender requests a web page via the HTTP protocol. The application layer hands this message down to the transport layer, which is running TCP. TCP cuts it into suitably large packets and prepends its header. The header indicates, which application layer protocol has sent the message to allow the receiver to handle the message correctly. In TCP, this demultiplexing key is termed a *port*. The current fragment number and the total number of fragments is indicated as well to allow the receiver to reassemble the packets correctly, even if they don't arrive in order[5]. The message is then passed down to the IP layer, where the IP address of source and destination host is added.

Below the network layer, in the data link and physical layers, a variety of protocols can be used. For home, campus and office networks, *Ethernet* is the most popular protocol below TCP/IP. It can handle a wide range of physical media, e.g. the IEEE 802.3 standard defines Ethernet on electrical wires or optical fibres and IEEE 802.11 defines Ethernet for wireless devices.

We will now describe the essential protocols of the suite in more detail.

Network Layer The most important protocol of the TCP/IP suite is the network layer protocol *IP*. IP is an acronym for *Internet protocol*. The current version is IPv4, but the next version, IPv6 [11], is already being deployed. All that IP provides is the delivery of small messages called *datagrams* from one host to another. This simplistic approach is largely responsible for the enormous adaptability of IP to various underlying network technologies. However, the IP protocol only offers a *best-effort* service, which means that it doesn't provide any guarantees about availability or performance of service.

Hosts communicating to each other via TCP/IP need an IP address. In the currently deployed version of IP, version 4, addresses are 32 bit integer numbers. Conceptually, an IP address consists of two parts[6]. The first part (higher order bits) forms the *network address* and determines to which of the many networks connected by the Internet it belongs. The rest of the address is called *host address* and determines to which of the hosts in the network it refers to. Where the network part ends and the host part begins is therefore also part of the addressing information and usually given in a bit mask called *netmask*. This distinction is essential to allow routers to efficiently guide packets in large internetworks since it allows hierarchical routing. Routers outside a network need only know whether the destination host is somewhere inside the network. The delivery to the final recipient is then the responsibility of the routers inside this network. We again refer to Figure 2.6 for an example. When routing a message from host 1 to host 2, router A applies the netmask with a bitwise AND to the destination address. The resulting IP address is then 10.3.0.0 and the router looks up the route for this so-called *prefix*. Appearently, router C

[5] Flow, congestion and other control data is also included, but not shown in the figure for simplicity.

[6] Actually, there are three parts: *Network, subnetwork* and *host* part. But we treat the network and subnetwork part as one to make the discussion more understandable.

with address 10.3.0.1 is responsible for all addresses in this network (10.3.0.1–10.3.255.255). Therefore, router A sends the message out on Interface I03. Router C then routes the message either directly to the subnetwork of host 2 or to the next network, depending on what is hidden inside the cloud.

Transport Layer Right above IP, one of the transport layer protocols, e.g., *TCP* or *UDP,* do their duty. TCP means *Transmission Control Protocol* and provides a reliable stream of data between two end points connected by a network running the IP protocol. UDP stands for *User Datagram Protocol* and provides a very slim message delivery service which in essence makes IP datagrams accessible to applications. Consequently, UDP is not more reliable a service than IP.

Application Layer The protocols in the application layer[7] provide the interface between the applications and the TCP/IP stack. The most popular are *HTTP,* which is used by the *World Wide Web* and *SMTP,* which is responsible for the delivery of e-mails.

2.1.6 Communication Modes

Similar to communication among human beings who have the ability to either speak to a crowd, talk with a group or whisper into each other's ears, computers adapt their modes of communication to their needs as well.

There are mainly three communication modes: *Unicast, broadcast* and *multicast.* While unicast and broadcast communication is used heavily in today's computer networks, multicast is not in very wide use, at least not between geographically dispersed computers. We will give possible reasons for this fact in Section 2.3.

Please note that unicast communication may be uni- or bi-directional whereas broadcast and multicast communication is always uni-directional.

Unicast Unicast is the standard mode: One host sends a message to a single designated host. The sender puts the address of the receiver into the destination field of the message header and the network delivers the message.

An example is shown in Figure 2.7: Host 1 sends a unicast message to host 2. This message goes from the sending host 3 via a switch to router A, then router D and router C. From there, it can directly reach host 2 via the subnetwork, passing 2 switches.

Broadcast In this mode, data is emitted from one host to all other hosts. In this context, the set of all hosts is constrained by physical or logical limits. For example, if a host connected to a switch sends a broadcast message, only hosts connected via switches will receive it. If a router is connected to the switch, the router will receive the message, but not forward it to avoid a misbehaving host to flood the whole network. Broadcast communication is for example used when host 1 needs to send a message to host 2 in the same network, but doesn't know the physical address of 2. The protocol responsible to resolve this dilemma is called *address resolution protocol (ARP)* and runs in the data link layer. It works like this: ARP sends a broadcast message asking "Who has this IP address?" Host 2 receives this request and sends a reply with its physical address. Host 1 now can address host 2 directly and all subsequent communication is unicast. Figure 2.8 shows a situation where in all three subnetworks one node is sending a broadcast message. All nodes in the same broadcast domain receive it, but routers ignore brodcasts to ensure that one misbehaving host cannot flood the whole Internet with useless broadcast messages.

[7] In contrast to the TCP/IP protocol stack, the OSI model divides the application layer into three distinct layers: *session, presentation* and *application* layer.

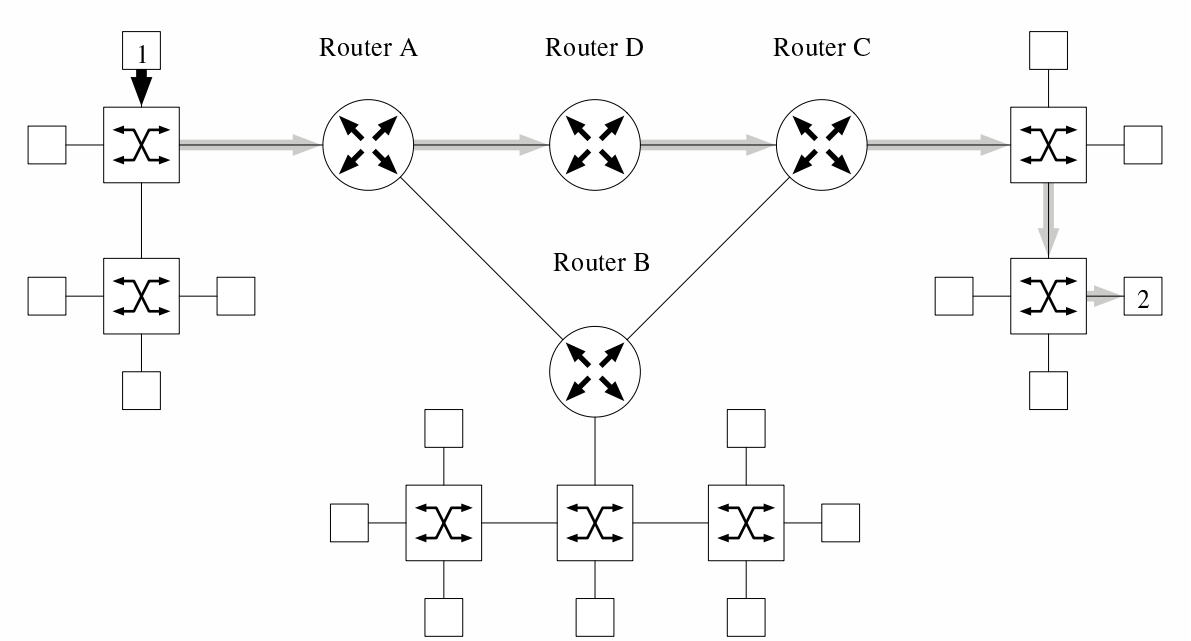


Figure 2.7: *Unicast communication from host 1 to host 2*

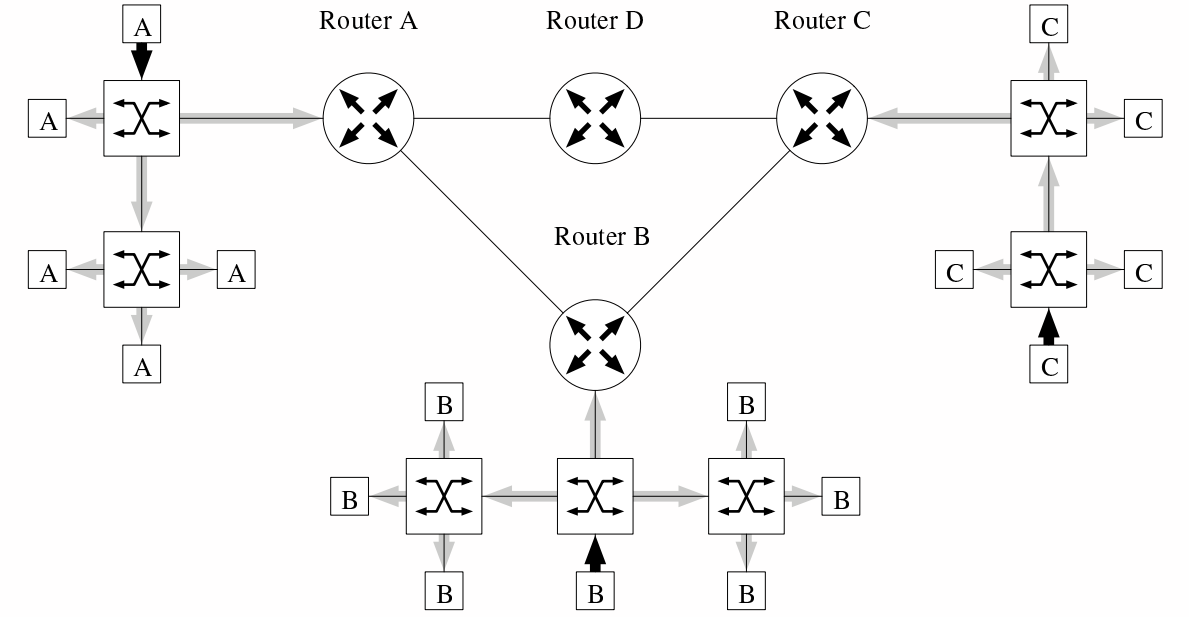


Figure 2.8: *Broadcast communication is confined to the broadcast domain, i.e., subnetworks A, B and C. The routers ignore broadcast messages.*

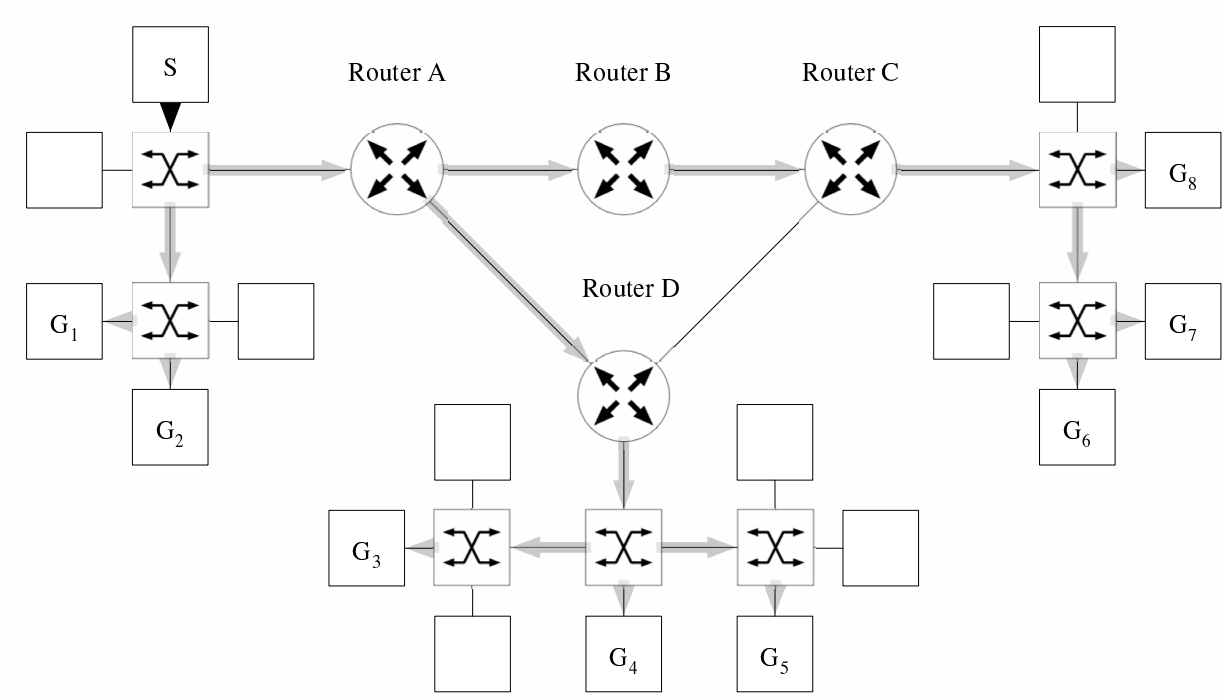


Figure 2.9: *Multicast communication from the mulitcast source host* S *to all subscribers, denoted by* $G_{1..8}$.

2.1.7 Multicast

The idea behind multicast is to let the network deliver messages destined at a group of hosts instead of sending a separate copy of the message to each and every receiver. Multicast communication seems to be situated between unicast and broadcast. For multicast communication, a set of participating hosts needs to be defined. This set is called a *multicast group*. When a member of this group sends, it is the *source*, and all other members are *receivers* or *subscribers*. Every multicast group needs to be assigned its own *group address* from the IP multicast class D network with the address range 224.0.0.0 through 239.255.255.255 as defined in [23].

In a packet-switched network with network layer support for multicast, the multicast delivery from one host to all other hosts of the group works in the following manner: The source sets the destination address of the message to the group address and inserts the message into the network. The network infrastructure, i.e., the routers, then forwards the message towards the receivers and replicates it as needed. With network layer support for multicast, the actual delivery is completely transparent to all group members. In Figure 2.9, it is shown, how a multicast message from the source host *S* is delivered to the group members denoted by $G_{1..8}$. The source host sends the message to its default gateway, router A, and the latter then handles the delivery[8]. A copy is sent to routers C and D, which replicate it and send copies to all group members in their network.

The multicast service can be implemented above the network layer as well, though, and such schemes are the subject of this thesis. Multicast protocols running above the network layer are called *application-layer* or *overlay multicast protocols* and are introduced in Section 2.4.

[8]Nodes on the same subnetwork receive the multicast message via link layer multicast addressing, but this is beyond the scope of this text.

2.2 Brief History of Internet Multicast

Multicast adddressing was defined in the IP protocol from the beginning as a separate address range. In its early days, multicast was confined to *local area networks (LANs)*, i.e., networks consisting of end systems and hubs using Ethernet or *Token Ring* technology. Those schemes inherently supported multicast since all packets were available to all hosts on a network. However, extended LANs connected through active devices like switches, and internetworks connected by routers were unable to deliver multicast packets.

Several proposals were published, e.g. [24], but the multicast era started only when Steve E. Deering introduced multicast extensions to the existing unicast routing infrastructure in the proceedings of the ACM SIGCOMM '88 conference [25], and in more detail in December 1991 in his Ph.D. thesis "Multicast Routing in a Datagram Internetwork" [26].

Deering's work resulted in the first global multicast network, the *MBone, [27], [28],* a set of multicast-capable networks connected through IP tunnels. The routing protocol was *Distance Vector Multicast Routing Protocol (DVMRP) [4]*. In March 1992, the MBone was comprising about 20 hosts. In an experiment, these machines successfully received a multicast audio stream from a meeting of the *Internet Engineering Task Force (IETF) [14]*. DVMRP assumes that most or all hosts on a multicast-enabled network wish to receive multicast traffic, thus it is considered a *dense mode* protocol. While this may have been at least partially true for the MBone, this assumption doesn't hold for today's multicast groups.

In large networks connected by routers supporting multicast in hardware, the problems of dense mode multicast need to be addressed. Currently, the most widely deployed *sparse mode* protocol according to [29] is *Protocol Independent Multicast (PIM-SM) [9]*. But even with later versions of PIM, a lot of IPv4-inherent problems remained unsolved and new challenges were introduced. The most critical open issue is the allocation of multicast addresses. IPv4 only offers a flat address space for multicast which doesn't give any hints as to where a multicast source or its subscribers are located. Another serious problem is the openness of the approach. With the current popularity of *Distributed Denial of Service (DDOS) [30]* attacks, the total lack of access control from traditional IPv4 multicast schemes poses a threat to all hosts in a multicast-enabled network.

The problems of address allocation and access control were recently addressed by schemes dedicated to single-source multicast applications. Protocols of this type include *Express [31]* and *Source Specific Multicast (SSM) [32]*.

There are a lot of other open issues and some of them will be discussed in the next section.

2.3 Motivation for Overlay Multicast

First of all we would like to address the question why we even think about implementing multicast services in the application layer, if they exist in the network layer as part of IPv4. Or in other words: Why did IPv4 multicast fail in that is has not been deployed Internet-wide?

There exist several lines of reasoning. One popular explanation is with the chicken and egg problem: The Internet service providers (ISPs) didn't deploy multicast—the egg—because nobody seemed to want to use it. The potential users—the chicken—didn't use it because it hadn't been deployed wide enough.

There are philosophical objections as well. The implementation of multicast services in the network layer violates a widely accepted design paradigm summarized in the paper *End-To-End Arguments In System Design* [33]. Applied to multicast, it says in essence:

> *Since multicast services cannot be provided completely without support of the application layer, they should only be implemented at a lower layer if the gain in performance is so large that it justifies the additional cost of more complexity in the lower layer.*

Another general network design paradigm is not in favour of the network layer either:

> *No service should be implemented in a particular layer unless this layer can completely and reliably implement the whole functionality.*

One very important function that is lacking from IPv4 multicast is address management: How are multicast addresses assigned and by what means are potential clients informed about the available multicast services? A third paradigm is more specific to the network layer:

> *The network layer should not contain any state information.*

While unicast routing has been successfully implemented in IPv4 and is doing an impressive job in routing across a subset of the several thousand *Autonomous Systems (ASs)*[9] comprising the Internet, the same is obviously not true for multicast. With IPv4 multicast, any router needs to potentially keep membership information of *all* hosts which are directly connected to it or to a dependent router. This limits the scalability when groups are dispersed among several ASs and consist of thousands of members.

While the above arguments may seem rather esoteric, there are a lot of tangible and technical problems with IPv4 multicast as well. We will summarize the most prevalent in the next section.

2.3.1 Problems of IPv4 Multicast

Accounting For Internet service providers and backbone owners, multicast opens a lot of difficult questions: How is the transferred data accounted for? Who pays the consumption of the precious resources of the routers used for multicast routing and group state information?

Interdomain Routing If a multicast group spans multiple autonomous systems, another problem arises: Since there are multiple protocols available and some ASs even offer no multicast at all, static IP tunnels need to be set up to interconnect disjoint multicast zones. This problem has lately been mitigated by the introduction of interdomain multicast routing protocols. However, a long-term solution has yet to be found.

Deployment Model For IPv4 multicast to fulfill its purpose, all, or close to all, hosts should be connected to the same multicast domain. This makes the deployment an “all or nothing” decision, and most providers went for nothing.

Access Control What’s more, for the streaming of copyrighted material, for multi-party games, database replication and distributed computing, IPv4 multicast often cannot be used because it provides neither authenticity nor confidentiality. Of course, the data might be encrypted at the application layer and then distributed via network layer multicast. But this would no longer be transparent to the application.

In order to work around all these problems at once, considerable effort has been put into the development of new multicast schemes at the application layer. Accounting, encryption, authentication—virtually all features lacking from IPv4 multicast can be provided with application layer multicast. Of course, this comes at a price. How steep this price is we will try to at least partially assess with our simulation experiments in Chapter 4.

2.4 Overlay Multicast Primer

As the term overlay implies, we are talking about a virtual network topology laid out on top of the underlying physical network. The term *application layer multicast* denotes a subset of schemes which are

[9] An autonomous system is a network or internetwork that is under the administrative control of a single entity. The term *routing domain* is often used when talking about routing between ASs which is then called *interdomain routing*.

running in the application layer. However, the two terms are used interchangeably because all application layer schemes inherently use an overlay network.

This virtual network can conceptually be divided into two structures: A redundant control topology termed *mesh* and a spanning tree called *data delivery tree*. Common to all schemes is also that they need a central entity to bootstrap the protocol. This important machine is often called *rendez-vous point (RP)*. However, this is no different in IPv4 multicast. The bootstrap information in this case is the group address, the distribution of which is a non-trivial problem for which a satisfactory solution has not been proposed yet. In overlay multicast schemes, the bootstrap information may also be offered by another out-of-band entity, e.g. a web site where interested recipients need to click on a link or copy and paste an address.

Routing in the application layer seems to be inherently less efficient. This becomes evident when we take into account the overhead experienced by packets travelling through the complete protocol stack at every node. In addition, whenever a tree is used for data distribution, all nodes with more than one child will transmit a separate copy of every data packet to every child, resulting in a manifold increase of upstream usage. Since the upstream bandwidth of current modem and broadband connections is usually small in comparison with the ever increasing downstream capacity, this increased stress is a major problem of overlay multicast schemes and needs to be minimized at any rate. The motivation behind overlay multiast schemes is not to improve the performance over native IPv4 protocols but to provide comparable performance without network infrastructure support.

2.4.1 Classification of Overlay Multicast Schemes

There exist several different classes of overlay multicast schemes. A widely accepted classification divides them into two main classes: *Host-based* and *replicator-based*.

Host-Based *Host-based multicast*uses only end systems to build a distribution overlay. Every node of the overlay acts potentially as a router and as a data source or sink at the same time. This is in contrast to IPv4 multicast, where the whole multicast routing algorithm is running on dedicated routers, completely transparent to the end systems.

Replicator-Based *Replicator-based* multicast needs infrastructure support in the form of dedicated hosts capable of running the multicast routing protocol to forward and replicate the data packets disseminated by the source host. From the point of view of end systems, this approach is very similar to native network layer multicast.

The overlay network on which the multicast routing algorithm constructs the data delivery tree may be established in several ways and allows to further classify the schemes according to [34].

Tree-First The most intuitive approach is called *tree-first approach*. Every new node requests a list of members from the rendez-vous point and subsqequently asks these members to be added to their list of children. Once a node has found its parent, it tries to evaluate other nodes and if it finds one with better properties, it becomes its child. At the same time, it collects a list of members in its vicinity. These redundant nodes form the mesh and are substituted for the parent in case this host fails or the tree is partitioned.

Schemes applying this procedure are most useful for applications where high bandwidth is more important than low latency. Protocols using this approach are *Yoid [35]* and *HMTP [36]*.

Mesh-First Doing the same steps the other way around works as well and this procedure is called *mesh-first approach.* With this method, every node first selects a subset of the end-to-end links to form a redundant mesh connecting all end systems either among each other, or with one or several replicators,

depending on the scheme. Using a subset of these links, a tree structure is then dynamically created to deliver the data packets.

Protocols of this class are efficient for small multicast groups, but do not scale well beyond a few tens of hosts. The most popular scheme of this class is *Narada [1].*

Implicit Performing both steps at the same time is also possible and is called *implicit approach.* Here, the nodes are usually arranged in clusters and the optimization works towards putting the nearest neighbors in the same cluster, while respecting upper and lower bounds of cluster size.

The advantage of this approach is its flexibility and scalability. *NICE [37], CAN-Multicast [38]* and *Scribe [39]* are popular representatives of this class.

2.4.2 Overlay Multicast Scheme Example

In Figure 2.10, an example of an overlay network is shown. The thick grey denote end-to-end connections which form part of the overlay network. These logical connections do not correspond to the physical links of the underlying network. Rather, they are virtual connections between end systems. In this example, host S is the multicast source and all hosts marked with $G_{1..8}$ are subscribers.

In the leftmost network, the two group members G_1 and G_2 can be reached by the source S using direct overlay links because they are in the same physical network. The receivers in the bottom network are connected to the source via the overlay link from S to G_4, which passes routers A and D. From G_4, data packets are delivered to hosts G_3 and G_5 via direct overlay links. The hosts in the rightmost network are attached to host G_4 via an overlay link passing routers D and C. From host G_8, data is delivered to hosts G_6 and G_7.

Note that in this example the physical link from router D to host G_4 is used twice for every data packet. The physical link from host G_4 to its switch is even passed by four copies of the same packet. This inefficiency is unavoidable because the overlay network is constructed without information about the physical topology. The overlay routing algorithm can measure certain properties of links, e.g. the roundtrip time of a packet, but it cannot determine the exact topology. Thus, every overlay network is inherently less efficient than the underlying physical network.

2.5 Aims and Goals of this Semester Project

The subject of the conceptual formulation for this semester project is

> *One approach to provide multicast services is application layer multicast, where data is routed via an application layer overlay to the group members. The goal of this thesis is to model and implement a generic simulation framework for such overlays.*

The complete document can be found in Appendix A. The question to be addressed is

> *What is the most efficient multicast service for wide deployment in the existing Internet among the existing alternatives?*

The procedure should follow these steps:

1. Study of related work.
2. Selection of a representative set of multicast schemes to put under study.
3. Discussion of the evaluation parameters and creation of application profiles.
4. Study the use of the OMNeT^{++} simulator.
5. Implementation of the chosen protocols in the OMNeT^{++} environment and measurement of the variables of interest.

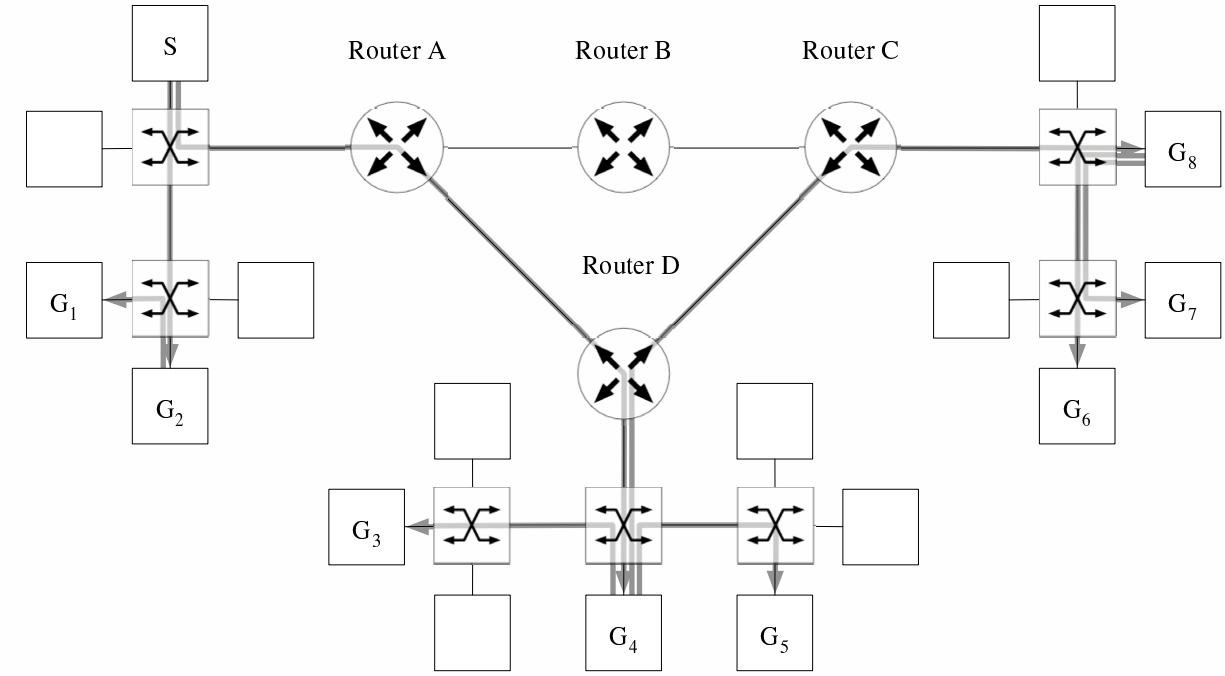


Figure 2.10: *Overlay multicast scheme. Here, the multicast forwarding is performed by ordinary end systems denoted by $G_{1..8}$. Note, that some of the physical links are used several times in contrast to the example in Figure 2.9 where the multicast delivery is handled by the routers.*

6. Processing of the results and derivation of conclusions.

To what extent these goals were achieved is discussed in Appendix B.

In the next chapter, we will present the schemes we considered for the evaluation and present our overlay multicast protocol.

Chapter 3

Schemes Under Study

This chapter discusses our selection of schemes to evaluate and a description of our custom overlay multicast protocol.

3.1 Scheme Classes

3.1.1 Native IPv4 Multicast

We have analyzed the most popular multicast protocols for IPv4. We rated them qualitatively against the following criteria: Current deployment status, flexibility and suitability for future Internet-wide deployment.

DVMRP The first multicast protocol proposed for TCP/IP networks in 1988 was the *Distance Vector Multicast Routing Protocol (DVMRP) [4]*. It is a multicast extension to the unicast distance vector routing protocol *Routing Information Protocol (RIP) [40], [41],* but it builds its own multicast routing table based on which it constructs a *reverse path forwarding* tree. When DVMRP was developed, it was assumed that almost everybody in a network would want to subscribe to a multicast group and it is accordingly termed a *dense mode* protocol. Data destined at multicast groups are sent to the designated DVMRP routers of all subnetworks in a DVMRP-enabled domain. If there are no multicast subscribers in a certain subnetwork, its designated router may ask its upstream router to be pruned from the tree. Both routers store this information for a few minutes and then a new prune message needs to be sent by the downstream router. This costs potentially a lot of memory in routers which are connected to many networks with no subscribers. Obviously, this approach does not scale well. Additionally, the maximal diameter of a DVMRP multicast group is limited to 32 links. To remedy this limitation, a hierarchical model was proposed in 1995 [42] to increase the scalability, but it was not blessed with long standing success.

DVMRP failed our examination because it is depending on one particular unicast routing protocol and does not appear to be scalable enough for the Internet of today.

MOSPF The unicast routing protocol that mainly replaced RIP was the link-state routing protocol *Open Shortest Path First (OSPF) [43], [44]*. Based on this, the *Multicast Open Shortest Path First (MOSPF) [5]* scheme was proposed in 1994. For multicast, the link-state updates are extended by group membership information. This allows all routers in a routing domain to draw a complete, up-to-date image of the topology and group membership. When a multicast data packet arrives at a router, this device computes a shortest-path *source-specific tree* rooted at the subnetwork of the sender. If this calculation shows that the router forms part of this tree, it forwards the packet accordingly. The computation of the shortest-path using Dijkstra's algorithm, however, is computationally involved and

the distribution of the link-state packets relies on a reliable broadcasting mechanism called *flooding*, which is not scalable for wide area networks like the Internet.

MOSPF thus failed also because it relies on a specific unicast routing protocol and due to strong concerns regarding its scalability.

PIM We have chosen *Protocol Independent Multicast (PIM) [9], [10].* as a representative protocol for IPv4-based multicast. PIM offers high flexibility as it provides two different modes of operation: For sessions with high node density, it may be run in dense mode (PIM-DM), whereas if density is low, it can be run in *sparse mode* (PIM-SM). PIM-DM uses a *shared tree*, i.e., the data delivery tree is rooted at one router and is the same for all source hosts. PIM-SM starts with a shared tree as well but has the ability to switch to a source-specific tree later on if this seems useful.

PIM appears to be the most widely-deployed IPv4 multicast protocol, according to for example [29]. To make the comparison as fair as possible, we considered only PIM-SM, because most overlay multicast schemes are targeted at sparse groups. Unfortunately, we did not have enough time to run simulation experiments with this protocol, but it will be discussed in detail in Section 3.2.

CBT Focused on scalability from the very beginning was the scheme *Core Based Trees (CBT) [6], [7],* a *sparse mode* protocol. It uses the same root node termed *core* for all sources and uses a more complex algorithm to construct a shared tree than PIM-SM.

CBT version 1 did not have much success, the incompatible version 2 is not widely deployed either and version 3 is currently an expired Internet-draft [8]. CBT was thus disregarded because of lacking deployment.

3.1.2 Host-Based Overlay Multicast

In this area, we have looked at *Yoid* [35] and *End System Multicast (ESM)* [1].

ESM The routing protocol of the ESM scheme, *Narada*, is quite sophisticated and seems to make good use of the processing power offered in today's end systems. We decided to take this scheme as a basis and design and implement a custom architecture for our measurements. For easier reference, we will call our scheme *Overlay Multicast Protocol* or *OMCP* . It will be specified in detail in Section 3.3 on page 23.

Yoid This scheme is host-based in its basic mode, but its performance may be enhanced by the installation of dedicated replicators at critical points in the Internet. Yoid uses a tree-first approach with clustering. A new member gets a number of currently active group members and asks the most suitable to be its parent. Since Yoid is not specifically designed to support classical multicast applications like streaming but also employs caching for file transfers, we decided to use the more generally-scoped Narada.

3.1.3 Replicator-Based Overlay Multicast

We evaluated *ALMI [45], OMNI [46], Overcast [47],* and *Scattercast [48].* Our qualitative examination of the above protocols led us to choose OMNI. ALMI uses a completely centralized approach, which sets it very far apart from the distributed nature of the other two candidates. The application-specific extensions of Overcast and Scattercast made the comparison with network layer multicast appear questionable. The goal of OMNI, on the other hand, is a minimal latency distribution tree using dedicated replicator nodes.

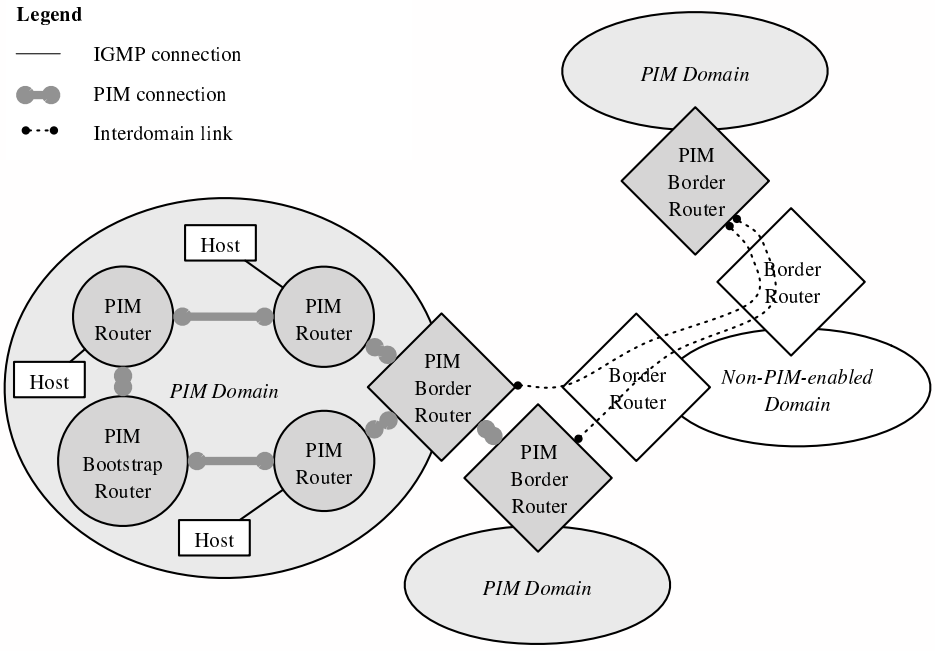


Figure 3.1: *Example internetwork topology to show how PIM operates across non-PIM-enabled domains*

3.2 Protocol Independent Multicast—Sparse Mode (PIM-SM)

PIM-SM version 1 is described in RFC 2362, published in June 1998 [9]. The most recent development was the submission of the latest PIM-SM version 2 specification to the IESG in April 2004 for consideration as a proposed standard.

All mentioned IPv4 multicast protocols rely on the *Internet Group Management Protocol (IGMP) [49]* to manage communication between end systems and their local multicast router. Hosts can join and leave groups by sending their router an *IGMP Join* or an *IGMP Leave* message, respectively. All hosts which have registered their membership in a group with a certain group address are then forwarded all packets destined to this address by the router.

PIM conceptually divides networks into PIM domains, i.e., areas with PIM support, and all other areas without support for PIM. Thus, the task of PIM is on the one hand to distribute data within PIM domains and on the other hand to provide for unicast connections using interdomain routing to connect these islands. An example of this architecture is shown in Figure 3.1.

The first IPv4 multicast protocol, DVMRP, assumed that all hosts in a network were group members and constructed a tree comprising all hosts. It then pruned branches where no group members were available. PIM-SM uses the opposite concept: It assumes, that no group members exist. Therefore, group subscribers need to send an *explicit join* packet to the rendez-vous point in order to start receiving data sent to the group. Group members need to check periodically whether they receive data. It is this explicit join model that makes PIM-SM so much more scalable than dense mode multicast protocols like DVMRP and MOSPF.

Another important difference is that PIM-SM uses a single tree to distribute data to all PIM routers with active group members. This tree is rooted at a well-defined machine called *rendez-vous point (RP).* This is in contrast to the *source-specific tree* model used by DVMRP where a source-specific tree is maintained for every group member that has sent some data recently. This increases the scalability tremendously. But using a single tree is not necessarily optimal for all sources in terms of latency, as we

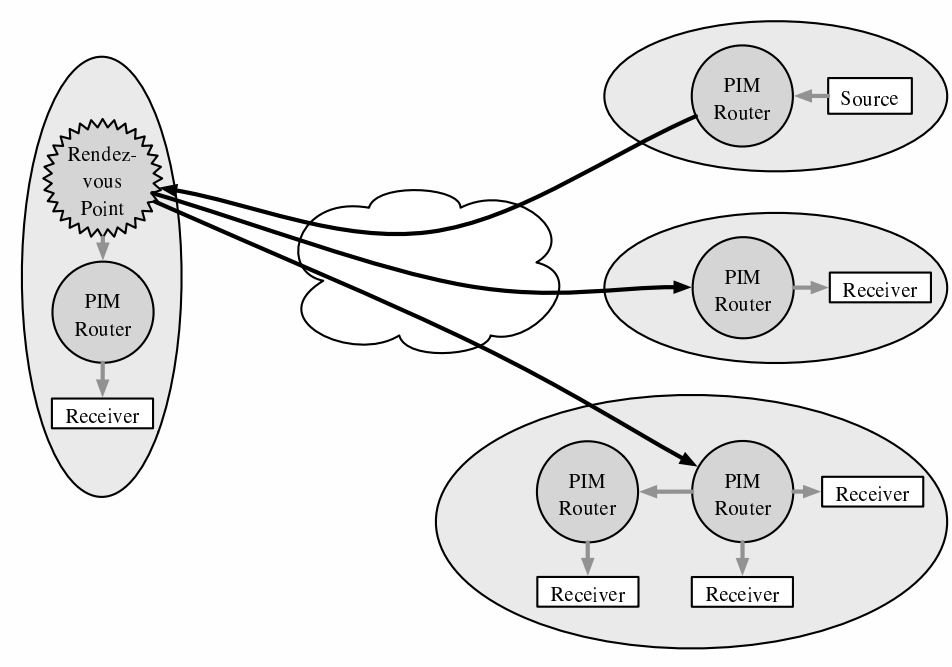


Figure 3.2: *PIM-SM shared tree. The shared tree is always available in PM-SM. The multicast source sends the data to the rendez-vous point, and from there, the delivery is performed along the same tree, independent of the source host. Note that all data packets from the source cross the rendez-vous point. (Border routers are omitted in the figure for simplicity.)*

will see later in an example. The selection of an optimal rendez-vous point is an NP-complete problem and is in all practical implementations approximated using heuristics.

All PIM routers who need to receive data for a certain group register their group membership at the rendez-vous point of this group. A rendez-vous point may serve several groups and every group of a particular domain uses only one RP. Information about RPs is distributed by *bootstrap routers* within a PIM domain. Every physical network needs at least one PIM router. This machine constantly collects information about rendez-vous points. PIM domains are connected via *multicast boundary routers* which serve as gateways and transmit information about rendez-vous points to the PIM domain at the other end. Every PIM router with active subscribers periodically sends a PIM Join data unit to the rendez-vous point to indicate that it still needs the packets for this group. The join process in a PIM domain works as follows:

1. The joining host sends an IGMP Join message to the designated router of its subnetwork.
2. The designated router searches its records about rendez-vous points for the responsible RP. Obviously, for this to be successful, the router needs to have received this information from a bootstrap router beforehand.
3. Multicast data packets sent from the source to the rendez-vous point are replicated at the RP and forwarded to all PIM domains with active group members. The border routers of these domains then forward data along the distribution tree to all PIM routers inside the domain which have sent a PIM Join message recently enough.

To put away with the inefficiency of a shared tree, PIM-SM establishes a source-specific tree once the data rate of a source exceeds a certain threshold. A comparison between a shared tree (dashed lines) and the corresponding source-specific tree is shown in Figure 3.2. The process that leads to such a tree is complex and beyond the scope of this text. It is remarkable, however, that this tree is established as soft state, which means that it is destroyed if the forwarding state has not been renewed during a given

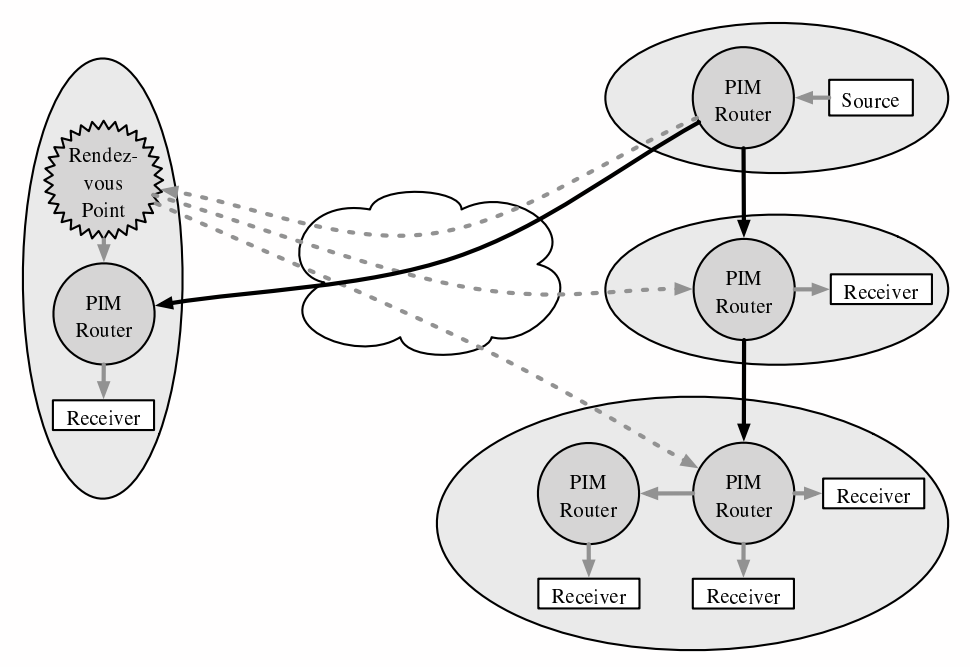


Figure 3.3: *PIM-SM source-specific tree. Such a tree is only established once the data rate of a source has exceeded a certain threshold. Note that this tree does* not *involve the rendez-vous point. The shared tree is drawn with dashed lines for comparison. (Border routers are omitted in the figure for simplicity.)*

timeout interval. This simplifies the protocol, but can lead to a high amount of control traffic, especially if large networks are involved, because receivers will try to keep their trees established even if no data are sent for an arbitrary period of time.

3.3 Overlay Multicast Protocol (OMCP)

In this section, we will describe Narada and point out what we have done differently in our custom protocol termed *OMCP* for easier reference. The scheme currently works towards minimizing the perceived overlay latency, but other metrics could also be used as long as they are attainable by end systems.

3.3.1 Contributions in OMCP

There are a number of differences between OMCP and Narada. We will explain them in detail in the description in the next sections, but here is a list of the most notable contributions:

- Negotiated improvement of the mesh overlay structure.
- Route poisoning to control the data delivery tree.
- Triggered routing updates for faster routing convergence.
- Faster incorporation of fresh members into the data delivery tree.
- More accurate measurement of link latency.

3.3.2 Mesh

Since Narada is a mesh-first approach, its performance is governed mainly by the quality of the mesh. A sophisticated mesh setup and improvement strategy is thus vital to the success of the whole scheme. The

mesh improvement should be based on the metric which is most critical for the application because the data delivery tree can at most perform as good as the mesh.

Mesh Establishment And Maintenance

The evolution of the mesh is not described in the Narada paper. In OMCP, nodes strive very fast towards their minimal degree and then become more selective about which links they add. It is therefore wise not to choose too high a minimal degree parameter, i.e., no greater than about five, to avoid adding lots of underperforming links which will be dropped later. Here is a short breakdown of how a multicast group evolves:

1. A node decides to initiate a group and makes the group address available to other potential members. A well-defined rendez-vous point could be put into service and keep track of active groups and provide a partial list of active members. Or there might be a web page, where groups and members can be registered and retrieved by new users.
2. To join the mesh, a node sends an *AddMeshLink* message to one or more active group members. The recipients will then decide based on their current number of mesh links if they can take one more mesh link. If yes, they send an affirmative reply and the mesh link is established. If not, a list of active members is sent back and the node chooses another member.

As soon as a node has established the first mesh link, it starts exchanging the following messages with its mesh neighbors:

- *Refresh messages:* These contain a list of all known group members and a sequence number for every member. If a node stops receiving refresh messages with increasing sequence number from a certain member, it assumes either the member to be dead or the mesh to be partitioned and starts the probing mechanism (see below).
- *Routing updates:* With these packets, mesh neighbors exchange their complete routing tables. The route entries contain the next hop address, the associated latency and the complete path to the destination. Routing updates are sent periodically. Additionally, an update is triggered whenever changes to the routing table occur and no RoutingUpdate is scheduled within a very short timeframe.
- *Pings:* Similar to ICMP echo requests[1], these messages request an echo message and are used to measure the latency of links. Depending on the application, those packets might be enlarged using padding to get more accurate measurements, e.g. for downloads of large files. Periodically, every member sends Pings to every member to make sure it has recent enough latency measurements.
- *Routing update requests:* Similar to Pings, this kind of message asks the receiver to reply with its complete routing table. The sender then uses this information to calculate if a mesh link should be added.

Mesh Performance Measurement

Only on the basis of recent and accurate latency measurements is it possible to improve the mesh constantly and to reliably find better links. In the specification of Narada, it is just mentioned that members periodically probe each other to measure the unicast latency.

Narada uses the same kind of messages for the measurement of unicast links and to check if a member is dead and requires the receiver to respond to these messages with its complete routing table. Since the size of the routing table may vary depending on the age of a node and the number of members in the

[1] *Internet Control Message Protocol (ICMP)* is a companion protocol to IP and is used for the communication between routers and end systems. *Echo Request* messages are mostly used to check if a host is running or to measure the roundtrip delay.

group, affecting directly the processing time at the receiver as well as the transmission time of the reply, we introduced three different packet types. Ping packets are used to measure the performance and may be of arbitrary length. It might be advisable to adapt the length of the Ping messages to the length of data messages. The second packet type is called *Probe* and is used solely to determine if a member is dead. The third kind is termed *RoutingRequest* and asks the receiver to send back its complete routing table.

We developed the following measurement mechanism for OMCP: Whenever a node is informed about a new member, it waits for a random time bounded by a parameter value and then sends the first Ping message to the new member. This random delay is essential to avoid that members which have just come online are flooded with Pings by all current group members. In steady-state, every node sends Pings to group members periodically. In Narada, the target is selected randomly. We think that this can be dangerous because the law of great numbers does not necessarily apply for groups with a few dozens of members. A node might send Pings unevenly distributed over all group members and thus never get latency information about potentially very fast links.

Our approach calculates the interval between two Pings to the same host, T_{ping}, with respect to a time parameter $T_{pingTarget}$ as follows:

$$T_{ping} = \frac{T_{pingTarget}}{N},$$

where N is the number of group members that the host is aware of.

Every T_{ping} seconds, an OMCP node does the following: It searches for the member record that has the oldest Ping time stamp, sends a Ping to this member and updates the Ping time stamp in the member record.

Under steady-state conditions, this approach guarantees, that a Ping message is sent to every group member in an interval of $T_{pingTarget}$. Thus, the latency measurement of no link grows older than $T_{pingTarget}$ and all group members are sent Pings in equal intervals.

The links to mesh neighbors are evaluated more frequently. This allows to detect almost immediately when a mesh link goes down and to replace it.

Mesh Improvement

The Narada paper gives a description of the procedure to evaluate if a new link should be added or a current one be dropped. The scheduling algorithm for the evaluation, however, is not specified. So we came up with the following approach in OMCP: Every node alternates between evaluating its current mesh links and checking out new links. If it decides to add or drop a mesh link, it subsequently waits for some time before running the next evaluation cycle to allow the routing to converge. For similar reasons, a fresh link is granted a grace period, during which it cannot be dropped.

Mesh Reduction Based on its own routing table and the stored routing tables of all known members, every node periodically calculates a value called *consensus cost* for every mesh link. The consensus cost of a link indicates, what the cost is of dropping this link in the view of the hosts at both ends. First, the nodes at both ends count, how many of their routes use the other host as first hop. This number is the cost from their point of view. The consensus cost is defined as the maximum of both these numbers. How this value is computed in OMCP is described later in this section.

It is not clearly defined whether Narada evaluates all links or only links with certain characteristics. In our protocol, we evaluate links with the following properties:

- The link is active.

- The link is not in use by the data delivery tree.
- The grace period of the link is over.
- The latency measurement and the stored routing table of the member at the other end are recent enough.
- The link latency is greater than a minimum parameter.

As specified in Narada, member A calculates the consensus cost to mesh neighbor B as follows: It counts the number of routes in its routing table that use B as first hop and stores this value. It then calculates the cost of dropping the link from member B's point of view using its stored routing tables in the member record and stores this value as well. The consensus cost is obtained as the maximum of both costs.

After the consensus costs have been calculated, the lowest cost link is dropped if and only if the cost is below the current *cost threshold.* This value depends on the number of mesh links the node has and on the node's perception of group size. How the cost threshold is computed is not defined in the Narada paper, though. We developed the following calculus: We denote the current number of mesh links with M. Let M_{min} be a parameter for the minimal and M_{max} for the maximal number of mesh neighbors. We define the range r as

$$r := M_{max} - M_{min}$$

and the position p as

$$p := \frac{M - M_{min}}{r}.$$

Let M_{target} be the optimal position between M_{min} and M_{max}. The number of group members in the perception of the current node is denoted by N. Since the maximal cost equals $N - 1$, we made the cost threshold $c_{threshold}$ depend on this variable. It is defined as

$$c_{threshold} = p * (1 - M_{target}) * (N - 1).$$

Furthermore, links which have a very low relative latency, for example, less than one tenth of the average latency of all mesh links, are never dropped.

To avoid loss of data packets, we added the following negotiation method: Mesh neighbors ask their peer if they may drop the link between them. Permission to drop the link is granted only under the following conditions:

- The number of mesh links is greater than the minimal degree parameter.
- The link is not in use by the data delivery tree.

To avoid packet loss when links are dropped, Narada requires nodes to continue sending data packets via dropped links for a transient time. We went one step further and added a function which actively vacates mesh links. This function is called when a mesh link should be dropped.

Vacating Mesh Links Whenever a mesh link needs to be dropped, in our approach, the node which wants to drop the link first vacates it as follows: It sends the other end a *VacateMeshLink* message, poisons all its routes which contain the other node and sends a RoutingUpdate to all its neighbors immediately. The node at the far end does the same. After some time, both nodes check if they still have routes using the link and if not, they ask their peer if it is ready to drop the link as well. If not, they wait again. After a maximal wait time, if the link could not be dropped, it is re-enabled because appearently the situation has changed and the link now appears to be necessary for the mesh.

Mesh Enhancement To evaluate if a mesh link to another member should be added, an OMCP node first sends the candidate a *RoutingRequest* message. This asks the recipient to reply with its complete routing table. Based on its own routing table and the one of the candidate, the node then calculate a value called *utility*, which indicates to what extent the addition of this link would reduce the latencies to all group members. As specified in the ESM paper, to compute the utility of adding a link to member B, node A does the following: For all group members, it calculates the new latency L_{new} if the link were added. If L_{new} is less than the current latency $L_{current}$, it increases the utility value u as follows:

$$u' = u + \frac{L_{current} - L_{new}}{L_{current}},$$
$$u = u'.$$

The link is only added, if the utility is greater than the current utility threshold. This threshold is not specified for Narada, it is only said that the value is a function of the degree of both members and the group size and that the link might also be added if the current overlay delay is very high and the new latency to the candidate would be very low. We used an approach based on the same variables as defined for dropping mesh links. The utility threshold $u_{threshold}$ is thus:

$$u_{threshold} = p * (1 - t) * (N - 1).$$

If the new link would reduce the latency to the candidate below a very low threshold, for example, lower than one tenth of the average latency of all mesh links, it is added regardless of its utility. The reasoning behind this is: If more links with minimal latency are part of the mesh, links with high latency will be dropped earlier and this leads to a mesh with lower overall latency.

Repair

Nodes can detect when group members die or the mesh is partitioned via the refresh mechanism. All mesh neighbors are required to periodically exchange their knowledge about other group members in *Refresh* messages which consist of pairs of member addresses and sequence numbers. If a node stops receiving increasing sequence numbers from a group member, it sets its status to *StaleMember* and puts its member record into the *stale queue* Q_{stale}. Periodically, the node checks if there are members in the stale queue. If there are any, it pops the first entry and sends a number of *Probe* packets to the concerned node. If it responds, its status is reset to *FreshMember*. If, however, no replies are received, the member is declared a *ZombieMember*. Should a Refresh message about a member with ZombieStatus arrive, it will be ignored. After some time, when the information about the disappearance of this member has spread over the mesh, all records pertaining to it are removed.

The scheduling is the same as in Narada: The above procedure is run periodically and is repeated until all members which have been in the stale queue longer than some period of time have been processed and with probability $P_{continue} = \frac{|Q_{stale}|}{N}$, one more stale member is probed.

In Narada, it is said that all mesh neighbors jointly determine if a node has failed and only then propagate this information through the mesh. However, how this propagation works, is not clear. One possibility would be via refresh messages, for example by resetting the sequence number to an invalid value.

Since preliminary tests showed that it is vital for the reliable delivery of the data packets to distribute the information about a node failure fast, our approach is more aggressive: As soon as a single mesh neighbor has sent a number of probes to the node and not received an answer, it perceives it to be dead. It sends an *Obituary* message to all its mesh neighbors. A member which receives an Obituary about a node checks its record of this node. If it is said to be an active member, it sets its status to ZombieMember and forwards the Obituary to all mesh neighbors except the one where it came from. If, however, it doesn't have a member record of the appearently dead node or if its record indicates that this node is not in the

active state, it ignores the Obituary entirely. This ensures that an Obituary concerning a particular host is forwarded at most once by any node and thus obituaries die after some time.

3.3.3 Overlay Routing

Like Narada, we run a distance vector routing protocol with path information on top of the mesh in OMCP. Mesh neighbors periodically exchange their complete routing tables. Additionally, as described in Section 3.3.2, nodes may ask for the routing table of a member that is not currently a mesh neighbor when it is considering the addition of a link.

For better control of the addtition and especially the replacement of routes, we introduce a poisoning mechanism. This allows a node to poison its routes to force itself to replace them. Additionally, through poisoned routing entries in RoutingUpdate messages, members can advise each other to drop routes which they consider invalid or which use mesh links they want to drop.

To allow for faster convergence, OMCP disseminates a *triggered* RoutingUpdate whenever a route's next hop changes, when a route is poisoned or when a mesh link is vacated. To avoid oscillation and too much routing information traffic, we use the following mechanism: If the routing table has been modified, no RoutingUpdate is sent immediately. Instead, a self message is scheduled to be received after a short period of time, e.g. 500ms. Only when this message is received, the RoutingUpdate will be sent. If, however, another routing change occurs, the node first checks if there is a self message scheduled already. If this is the case, it just waits because this message will cause the RoutingUpdate to be sent soon enough. With this algorithm, a minimal interval between updates is guaranteed.

3.3.4 Data Delivery Tree

Data is sent over a tree-like structure which is dynamically established for every group member as soon as it starts sending data. To be able to distinguish data packets, we use a sequence number mechanism. For greater fault tolerance, every node maintains a list of *tree parents* and *tree children* and relies upon this information in addition to its routing table when sending or forwarding data packets. This information enables nodes to recognize and accept data packets from former parents and send or forward to former children during a transient period. Additionally, fresh members or members which for some other reason have incomplete routing tables are also sent and forwarded packets. This allows fresh group members to participate in the delivery tree even though they do not have a complete routing table yet.

Sending Data Packets

The source sends the data packet to all its mesh neighbors which have at least one of the following properties:

- The stored routing table of the neighbor indicates that it can reach the sender in one hop.
- A copy of the neighbor's routing table is not yet available or it indicates, that the neighbor doesn't currently have a route to the source.

Forwarding Data Packets

The list of parents is used to decide if a data packet received should be forwarded. The decision process works as follows:

- The packet is dropped immediately if
  - The sender is not a mesh neighbor.
  - The packet is destined at a group which the node is not a member of.
- The packet is considered valid if and only if at least one of the following conditions holds:

  - The sender of the packet is the node's next hop to the source or the source itself. This is the base case for reverse path multicast. If the sender is not yet in the parents list, it is added now.
  - The node doesn't have a route to the source of the packet. If the sender is not yet in the parents list, it is added now.
  - The sender has been the node's parent until recently and the transient accept period is not over yet.

- If the sequence number is greater than the last, the node consumes the packet and forwards copies to all neighbors for which any of the following applies:
  - The stored routing table of the neighbor indicates that the node is its next hop to the source.
  - The neighbor has been a child of the node until recently and the transient forward period is not over yet.
  - A copy of the neighbor's routing table is not yet available or it indicates that the neighbor doesn't currently have a route to the source.

3.3.5 Group Dynamics

Multicast schemes based solely on end systems are inherently more prone to node failures than router- or replicator-based environments. Since not only failures or misconfiguration but also intentional behaviour of users may cause nodes to disappear, appropriate counter measures need to be put in place to handle situations like member leave, service degradation and—most challenging—sudden node failure.

Member Leave

A member leave should not lead to any data packet loss since it can occur arbitrarily frequently. As described in the Narada paper, a node will forward data packets for a period of time after it has quit. A member wishing to leave the group first poisons all its routes and thus will only advertise unreasonably high costs causing its mesh neighbors to look for other routes. In addition to this measure, leaving OMCP nodes will ask their neighbors to vacate their mesh links, as described in Section 3.3.2.

But with end users operating the routing entities, it is not guaranteed that a leaving user will allow its computer to forward long enough, especially if they left because they were not satisfied with the performance of the service or because they need the bandwidth for something else.

The method used by Narada to detect node failures is based on the refresh mechanism and described in Section 3.3.2. In OMCP, members may detect a mesh link failure much faster when a Ping message they sent to a neighbor is not returned.

3.3.6 Further Improvement

Due to time constraints, we were unable to implement all the features we had in mind. Some of the more interesting ones are the following:

Multi-Path Routing During preliminary tests, we noticed that almost all data packet loss was caused by abrupt node failure. A multi-path routing scheme should handle such situations much better. Members would accumulate at least one backup route to every mesh neighbor. It can easily be shown, that even with this primitive multi-path scheme it would be possible to handle node failures without any packet loss after the failure has been detected as long as each member failure is detected before another member fails.

Use All Control Traffic to Measure Mesh Performance As mentioned before, the active monitoring of the state of mesh links is critical to the performance of the data delivery tree. Therefore, we propose using Refresh and RoutingUpdate packets to measure the roundtrip delay of mesh links. Since these messages need a non-negligible time of processing at the receiver, they should be time stamped first when entering and second when leaving the OMCP application. The sender could then get an accurate measurement by subtracting this processing time from the perceived roundtrip time.

Harvest Topological Information of the Underlying Network In addition to the experimental deduction of the structure of the physical network, nodes might use their network address and netmask to recognize members on the same subnet. Additional information could be gained using freely available services like *DNS reverse lookup* or *whois lookup*. Furhermore, it might prove to be helpful to use a strategy similar to the UNIX traceroute utility in order to reveal more of the physical substrate on top of which the overlay topology is running. This approach would allow to make use of network layer information for multicast without causing any additional memory in routers to be wasted.

Accumulate Control Messages In contrast to the experiments conducted by the developers of Narada, we assumed that node failures were quite common and therefore used much shorter intervals between control messages. It makes no sense, however, to for instance send one RoutingUpdate message followed immediately by a Refresh message, since the conveyed information could just as well be put into one packet. What's more, Pings could be replaced by data packets during periods of high data rates with a bit in the data packet which indicates to the receiver that it should immediately send a Ping reply to the sender.

In the next chapter, we will present the specification of the application profiles we developed for the simulation experiments and specify the parameters to be measured as well as the heuristics we will use for the evaluation.

Chapter 4

Model and Evaluation Method

To compare several multicast schemes in a way that is at the same time fair and reproducible as well as meaningful for practical applications is a daunting task. We decided to rate the different approaches with respect to four application profiles. In this chapter, we introduce first the constituting criteria, i.e., metrics and heuristics, and then the profiles. Due to our limited time budget, we could only implement and evaluate our own overlay multicast scheme. We term this scheme *OMCP* for easier reference.

4.1 Methodology

In order to allow for a comprehensible comparison, we decided to base our evaluation on measurements of key performance characteristics. The success of a certain protocol in terms of popularity, however, does not depend solely on hard facts. Therefore, we also include heuristic criteria in our evaluation to account for differences in complexity of the implementation and other characteristics.

4.1.1 Metrics

While our simulation framework allows to measure several metrics, we focus on the following three: stress, stretch and delay. All metrics may vary over the course of a session, i.e., a period of time where the data source is sending constantly. Hence, we mainly use averages over time and over all receivers for the evaluation.

Notation

We will use the following notation.

Number of physical links on unicast routes among group members	n_{links}
Duration of the session in seconds	S_s
Duration of the session in packets	S_p
Unicast delay from host i to host j in seconds	$d_{i,j}$
Multicast delay from host i to host j in seconds	$D_{i,j}$
Number of copies of a packet carried by the link between hosts i and j	$N_{i,j}$
Arithmetic mean of realisations X_k of a random variable X	$\bar{X} := \frac{1}{n}\sum_k X_k$
Statistical variance of realisations X_k of a random variable X	$\tilde{X} := \mathrm{VAR}_k[X_k]$

When we talk about the value of a metric M for a particular data packet, we denote this by a superscript: M^k.

Stress

This metric is defined as the number of identical copies of a particular data packet carried by a single physical link. For the data packet with sequence number k on the link between two devices (routers or end systems) i and j, stress is defined as

$$N_{i,j}^k.$$

Specifically, we are interested in the average stress of a particular link l between devices i and j over an entire session:

$$\bar{N}_l := \frac{1}{S_p} \sum_{k=1}^{S_p} N_{i,j}^k,$$

and accordingly its variance among all links

$$\tilde{N} := \underset{l}{\mathrm{VAR}}\, \bar{N}_l$$

For the evalutation, we use the mean of the average stress of all links

$$\bar{N} := \frac{1}{n_{links}} \sum_{l=1}^{n_{links}} \bar{N}_l.$$

During the setup phase or in transient conditions, stress may be very large. This is especially true for protocols which have some or all neighbors transmit data packets to hosts that have indicated that they have lost their parent. Thus, another indicative property is the maximal stress that has occured on any link from source i to any subscriber j due to any data packet with sequence number k, as given by

$$\max_k \max_{j,\, j \neq i} N_{i,j}^k.$$

Delay

This is the overlay data delivery delay from the source host i to a receiver j. Again, we are mostly interested in the the mean of the average delay to every receiver:

$$\bar{D} := \frac{1}{n} \sum_{j=1,\, j \neq i}^{n} \bar{D}_{i,j},$$

and the variance among all subscribers

$$\underset{j}{\mathrm{VAR}}[\bar{D}_{i,j}].$$

Stretch

This metric measures the delay overhead incurred by overlay multicast compared to the respective unicast delay. The mean stretch factor for host j, when host i is the source, is denoted by

$$\bar{K}_{i,j} := \frac{\bar{D}_{i,j}}{\bar{d}_{i,j}}.$$

In an optimal multicast scheme, the latency from any source to any subscriber is equal to the unicast latency, resulting in a stretch factor of one. While this is theoretically possible to achieve with a network layer multicast scheme, it is practically impossible with application layer multicast. We are mostly interested in the mean of the average stretch factor over all hosts

$$\bar{K} := \frac{1}{n} \sum_{j=1,\, j \neq i}^{n} \bar{K}_{i,j},$$

Fault Tolerance

Most network layer multicast schemes do not focus on fault tolerance, as the reliability of the forwarding devices is usually sufficient. This is not the case with ordinary personal computers. Since most overlay multicast protocols are designed to run on such devices, their robustness against incidents like abrupt node failure and changes in network level routes plays an important role. Unfortunately, robustness is very difficult to measure. Because our time was limited, we decided to run all measurements in a setting where no network level route changes occur. All routes are computed before the simulation starts according to a shortest path algorithm and are static over the simulation run. To at least estimate the fault tolerance, we performed simulation experiments without node failures and again with a certain node failure probability. This is not a strict reliability measurement, but still can give a basic indication about the robustness of our scheme.

Further metrics

Due to time constraints, we could not perform all measurements we had planned. Here is a brief overview of what other metrics we deem important.

Efficiency Multicast protocols force routers or end systems to keep a considerable amount of state information about group membership etc. Further, they cause a considerable amount of additional control traffic. Both these quantities can be measured and compared in both classes of multicast protocols.

Packet Loss It is not necessarily the objective of a multicast protocol to guarantee reliable delivery of data packets. But in any case, packet loss should be avoided at all costs. For instance most streaming applications do not tolerate packet retransmissions. Packet loss is easily measurable. In our scheme, no packet loss occurs under the assumptions we made, but as soon as hosts fail, a nonnegligible number of packets is lost.

To assess the performance of a scheme in terms of packet loss, a similiar approach as proposed in the next section for the assessment of service interruptions might prove useful. A percentile-based evaluation [50] could also be used.

Setup Time The time delay between initiating the join process and the reception of the first multicast data packet may vary from less than a second to several tens of seconds among the discussed schemes. Thus, it is an important factor when a multicast protocol is evaluated.

Hold Time This is the period of time during which a former group member is supposed to continue forwarding packets to its neighbors. While this is zero in all mentioned network layer schemes, it can be up to a few dozens of seconds for application level protocols.

Jitter This is a metric that is mostly relevant for media streaming applications. It is an indicator of how constant the interarrival time of data packets is during the session as perceived by the receivers. There are several different definitions for jitter. One of the more intuitive definitions is the following: For a given host j, jitter of packets received from source i is defined as

$$J_{i,j} := \operatorname*{VAR}_k[D_k],$$

assuming that no packets are lost and that the packets arrive in sequence. To compare the performance of multicast schemes based on jitter, statistical methods like percentile measurement are required. Another possibility is proposed in the next section.

Further information on jitter and other metrics relevant for multicast services is provided in [51], [52], [53] and [54].

Fairness The variance of the arrival times of data packets among the multicast group is a metric for the fairness of a multicast scheme. This is crucial for real-time applications like stock market tickers and online bidding platforms. It is defined as the maximum of the differences of the arrival time at the first and the last receiver of a data packet k:

$$\max_j D_{i,j}^k - \min_j D_{i,j}^k.$$

Of particular importance is the maximum of the above value because it determines if a delay-critical application can be run with a given multicast scheme or not. It is given by

$$\max_k \left(\max_j D_{i,j}^k - \min_j D_{i,j}^k \right).$$

In the next section, a method for the assessment of service interruptions is given. Such an approach seems to be indicated to evaluate fairness as well.

Service Interruptions Any interruption of the reception of data packets hurts the performance of the multicast network. But depending on the application, different kinds of interruptions have very dissimilar impacts on the quality degradation perceived by the application. Hence, this property is crucial when deciding if a certain scheme is suitable for a particular application.

In the case of messaging, it doesn't hurt the performance, if every twentieth packet is lost, because this service by definition only uses very little bandwidth and lost packets can be retransmitted quickly. Download, in contrast, uses all available bandwidth, and thus, in the same network, packet loss would probably be higher and download performance would be degraded noticeably.

On the other hand, if it happens that the service is completely unavailable for a period of one minute every ten minutes, this only lowers the download bandwidth by ten percent, but can render instant delivery of messages infeasible.

Measuring this property is a difficult undertaking. It needs to be defined, by what a service interruption is constituted to distinguish it from the metric packet loss. One possibility is to use a 95^{th}-percentile scheme, but this neglects the duration of the longest five percent of interruptions. In the next section, a more accurate assessment scheme is presented.

4.1.2 Class-based Assessment

One of the main concerns about percentile-based measurement [50] is that it offers no insight about how bad the worst results were. For instance, if a multicast-based messaging scheme guarantees that 95 percent of all packets arrive within ten seconds, it matters if the other five percent arrive within twelve seconds or one minute. We propose to use several classes which are assigned different penalty factors. According to the number of occurrences and the penalty factor, a single value can then be calculated that accurately represents the suitability of a scheme for the requirements of an application.

We will use the metric service interruption as an example. First of all, we need to define a few time periods with a corresponding penalty factor. Let S_s denote the duration of the session in seconds. The penalty factor indicates, how often an interruption of a certain length may occur within a time period before the service quality becomes inacceptable. The number of occurences which render the service quality zero divided by the duration of the session, S_s, is the penalty factor.

However, the penalty factor for interruptions above the upper bound for the acceptable interruption duration, is *not* divided by S_s. This becomes more clear with an example. Let us consider an instant messaging service where service interruptions of 1 second or less do no harm, outages of less than ten seconds may happen three times per 100-seconds of a session and it must be guaranteed that all messages arrive within ten seconds. For such an application, the penalty factors are:

Duration [s]	Penalty Factor
$d_0 = 1$	$p_0 = 0$
$d_1 = 10$	$p_1 = \frac{100}{3} \cdot \frac{1}{S_s}$
$d_2 = 20$	$p_2 = 1$
$d_3 = 20$	$p_3 = 2$
$d_4 = 50$	$p_4 = 5$

According to this scheme, the service quality would then be calculated as follows. Let $n_{0..4}$ denote the number of occurrences of outages of the corresponding duration. The quality of the service is then defined as

$$Q := 1 - \sum_{i=0}^{4} p_i \cdot n_i \,.$$

If Q is equal to one, this incicates that the longest service interruptions were no longer than one second. When Q is $\frac{1}{3}$, this indicates that on average once per one hundred seconds of the session, an interruption of less than ten seconds occured. A value of $Q = 0$, however, indicates that at least one interruption has been longer than the allowed ten seconds or that more than three interruptions of between one and ten seconds have occured per hundred seconds of the session.

All negative values of Q correspond to an inacceptable service quality, either because an interruption of more than ten seconds has occured, or because too many interruptions of between one and ten seconds have happened. But even if the quality factor is negative, results are still meaningful and comparable among multicast schemes.

4.1.3 Heuristics

One of the key limitations of IPv4 multicast schemes is their limited scalability, even though—or perhaps because—they are quite simple. We give a rating for simplicity and scalability for both replicators (if applicable) and end systems.

Simplicity

We estimated, how easy it is to implement a protocol in hardware where memory is limited or in software where the processing of network packets is mostly limited in terms of speed.

Scalability

We analyzed the schemes for properties which limit scalability. We take into account, how much state information is necessary as well as what kind of processing has to be done to set up and improve the network (if applicable).

4.1.4 Application Profiles

Different multicast-based applications have different requirements in terms of quality of service. While it may do no harm to a audio streaming service if the delay from the source to the receiver is 500ms, such a delay can make another application, for example video conferencing, infeasible. This does not hurt when we consider software distribution or data backup services: If the multicast scheme offers a high average throughput over the course of an hour, this may work very well, even though the protocol might interrupt the service every five minutes for a few seconds to optimize the overlay topology.

Alas, we think that a fair comparison of different approaches to multicast communication is not feasible if only the requirements of one application are considered. Based on recent research papers and commercial multicast services, we came up with the following four application profiles. The quality of

service offered by our multicast scheme is compared to the needs of these applications. The results are given in the next chapter.

Streaming

Streaming is by far the most popular applications associated with multicast services. The media to be streamed might be a live broadcast of a conference or an on-demand broadcast of a university lecture. We assume that there is only one source, but a great number of receivers. The most important factor is stress. Stretch and delay are not as important because there is no interaction in this service. Another crucial property is jitter, but we did not have time to measure it and thus will not take it into account.

Messaging

With messaging, we mean collaborative work where several people send messages to all other group members, similar to instant messaging. Key criteria are low delay and stretch and high fault tolerance because such services are often used when people are far apart and possibly communicating via wireless networks. For purposes like stock market tickers, delay is crucial as well because if a message arrives too late, this might directly result in loss of money. Stress is not critical because the transmitted amount of data is assumed to be very small.

Download

More and more computer software and media is distributed over the Internet. Download is the classical example for replicator-based multicast. The infrastructure to allow huge amounts of data to be downloaded simultaneously by people around the world is already offered by Akamai *(EdgeSuite)* [55] and other companies. Download services can also be implemented using host-based multicast in a fashion similar to peer-to-peer file sharing [12], [13]. What matters here is bandwidth, i.e., low stress. Since we assume the data transfer to take longer than a few seconds, delay and stretch are not important at all.

Conferencing

A video conference of a group of people, dispersed around the globe, is probably the most demanding application for a multicast scheme. In contrast to streaming, all subscribers may also act as data sources. It has been shown in [51] that delays of more than 200ms render interaction almost infeasible. Hence, low stretch and low delay are of vital importance for this service. Jitter should be very low as well. Since the distances among the group members are assumed to be long, network outages need to be taken into account and high fault tolerance is necessary. The required bandwidth is large because every group member needs to receive a video stream from every other member. Thus, a scheme with low stress is more likely to provide enough netto throughput.

4.1.5 Weighting

Based on the above discussion of the application profiles, we set up an evaluation procedure as follows. The goal is to get a rating between zero and ten for every scheme with every application profile. Since our heuristic estimations are not as reliable as results of simulation experiments, we weighted them only with a factor of three, while measured metrics are weighted with seven:

$$7 \cdot \sum_{\text{metrics}} + \; 3 \cdot \sum_{\text{heuristics}} \; .$$

The weighting of the metrics and heuristics is laid out in a way that ensures that an ideal scheme gets a total of one point for the metrics as well as the heuristics. Actually, it was planned to take the results

of the network layer multicast protocol PIM as benchmark, but we did not have time to implement and test this protocol. Therefore, we use an ideal value of one for both the average stress, $\bar{N}_{ideal}$, and stretch, $\bar{K}_{ideal}$. The ideal value for the average overlay delay, $\bar{D}_{ideal}$, is the average of the unicast delay, $\bar{d}$.

To get the weighted value of, for instance, stretch for an application profile where stretch is weighted with a factor of 0.7, the following equation is used:

$$\bar{K}_{weighted} = 0.7 \cdot \frac{\bar{K}_{ideal}}{\bar{K}_{OMCP}}.$$

The weighting of the metrics and heuristics for all application profiles is given in Table 4.1.

Table 4.1: *Application profile weights of metrics and heuristics*

Application	Messaging	Download	Streaming	Conferencing
Key factors	Delay (Reliability)	Stress	Stress (Jitter)	Delay Stress (Jitter)
Metrics				
Stress		1	0.8	0.3
Delay	0.7			0.5
Stretch	0.3		0.2	0.2
Heuristics				
Simplicity	0.2	0.4	0.3	0.1
Scalability	0.8	0.6	0.7	0.9

4.2 Simulation Experiments

In this section, we specify the experiments we performed with our overlay multicast scheme, OMCP. The results and discussion are given in the next chapter.

4.2.1 Simulation Software

We based the implementation of our overlay multicast scheme on the *Discrete Event Simulator OMNeT++*. We have chosen this software over the standard, *NS 2 [56]*, for the following reasons: OMNeT++ appears to be easier to learn and provides a cleaner interface to the programmer. Additionally, it is constantly improving and has a supportive user community.

4.2.2 Network Topology

The simulation experiments are based on the *Transit-Stub Domain* model, an abstract representation of today's Internet created using the topology generator *GT-ITM [57]* of the Georgia Institute of Technology, available at [16].

We assume that there is no other traffic on the network. To accomodate for this assumption, we added an artificial queueing delay in every router. Thus, the queue sizes of the routers vary over time and influence end-to-end delays. All other properties of the network are constant over the course of a simulation run.

For our small-scale evaluation, we used only one topology. For a statistically sound evaluation, one would have to run the same experiments with many different topologies until the means of all measurements stabilize.

Backbone Network

The topology consists of 38 routers. The network comprises 12 stub domains and 6 transit domains. All 18 domains have an average of 3 interior routers. All routers have the same constant queueing delay, i.e., the time they require to process a packet is assumed to be constant and is the same for all routers. The links of the backbone have a bit error rate of zero. The delays of the links are distributed randomly, as well as the per-link capacity. They are chosen by GT-TIM in a way that ensures that packets to destinations in the same domain will take a route entirely within the domain.

The queueing delay was determined by evaluating the roundtrip times of three UDP packets sent to every router along the routes from the network of the ETH Zurich to the following hosts:

Switzerland www.bluewin.ch [195.186.6.80] and www.nine.ch [193.17.85.38]

USA www.berkeley.edu [169.229.131.109], www.apple.com [17.112.8.11] and www.lycos.com [213.140.50.210]

China www.shanghai.com [210.177.1.110]

Japan www.jal.co.jp [210.174.170.135]

This delivered the following results: The average number of hops per route was around 17 and the average roundtrip time was about 105ms. The roundtrip time incurred by one hop was 10ms on average. Thus, the queueing delay of a router seems to be approximately 5ms.

However, the average length of a path in our topology is only six hops, which corresponds rather to the European part of the Internet. To make the topology more similar to a a multicast group with some members in other parts of the world, we set a higher queueing delay of 10ms.

For comparison: The one-year average roundtrip time of ping packets sent from the ETH Zurich to the web server of the Massachusetts Institute of Technology in the USA is 96.5ms, which is equivalent to a unicast delay of 48.25ms. This value is quite constant over the course of a day. The average of twenty ping packets sent every five minues is always below 100ms.

The delay incurred by the physical link does not play an important role in our topology. It is 0.10294ms on average with a standard deviation of 0.047310.

End Systems

Every end system is connected to exactly one interior router of a stub domain and there are no two end systems connected to the same domain. The access links have a bandwidth of 1MBit/s and an error rate of zero. The delays are distributed randomly. The multicast group consists of one source host and 10 subscribers.

4.2.3 Scenario

For the evaluation of OMCP according to the application profiles, we used a very optimistic setting where no nodes fail. Further, we have run many different scenarios where nodes fail with a certain probability and within certain time intervals. We give a few informal results of such settings in the next chapter. In this section, we will first describe the most important parameters and then specify the scenario we used for the evaluation. All parameters of the simulation which are not discussed here are explained in Appendix C.

Activity of the End Systems

The end systems calculate the following points in time at the beginning of a simulation run according to a uniform distribution:

Join Probability P_{join} Probability, that a node will join the group.
Parameter: `d_join_probability`.

Join Time T_{join} Time when a node begins to accept packets from other nodes and initiates the bootstrap process.
Interval: [`t_join_time_begin`, `t_join_time_end`].

Leave Probability P_{leave} Probability, that a node will leave the group.
Parameter: `d_leave_probability`.

Leave Time T_{leave} Time when a node initiates the leave routine and informs its neighbors about its intention to leave the group. Interval: [`t_leave_time_begin`, `t_leave_time_end`].

Death Probability P_{death} Probability, that a node will die.
Parameter: `d_suicide_probability`.

Death Time T_{death} Time when a node fails entirely. It immediately stops sending and processing any packets.
Interval: [`t_suicide_time_begin`, `t_suicide_time_end`].

The source host additionally takes the following three parameters.

Data Start Time $T_{dataStart}$ Time when the source host begins to send data.
Parameter: `t_data_start`

Data Stop Time $T_{dataStop}$ Time when the source host stops sending data.
Parameter: `t_data_end`.

Data Interval $T_{dataInterval}$ Time between two data packets sent by the source host.
Parameter: `t_data_interval`.

Session

A session is the period of time when the data source is sending data packets. From the subscriber's point of view, this is an unsuitable definition and we use a more precise one. There are three parameters governing when a seesion begins and ends in the notion of end systems. During a session, all nodes are expected to receive all data packets, otherwise packet loss occurs. Packets lost before or after the session are not counted.

Setup Delay D_{setup} Period of time during which a node is assumed to be in the process of joining the multicast group and thus is not expected to receive data packets.
Parameter: `t_data_setup`

Propagation Delay $D_{propagation}$ Delay between source and subscriber. This parameter is used to calculate the sequence number of the first packet which should be received by a subscriber.
Parameter: `t_data_propagate`

Packet loss is accounted using sequence numbers. The source numbers all data packets, starting at one.

Session Start

There are two cases which need to be distinguished. In the first case, the subscriber has joined the group at least D_{setup} seconds before $T_{dataStart}$ and the first data packets are expected at $T_{dataStart} + D_{propagation}$. Thus, the first expected sequence number in this case is 1.

In the other case, however, the first data packets are expected at $T_{join} + D_{setup} + D_{propagation}$. The first expected sequence number is the first packet sent by the source after $T_{join} + D_{setup}$ and is calculated as follows:

$$\frac{T_{join} + D_{setup} - T_{dataStart}}{T_{dataInterval}}$$

Session End

Again, there are two cases to be distinguished. If the subscriber stays in the group for at least $D_{propagation}$ seconds after the source has stopped sending, data packets are expected until $T_{dataStop} + D_{propagation}$ and the last expected sequence number is

$$\frac{T_{dataStop} - D_{propagation} - T_{dataStart}}{T_{dataInterval}}.$$

In the other case, data packets are expected until T_{leave} and the last expected sequence number is

$$\frac{T_{leave} - D_{propagation} - T_{dataStart}}{T_{dataInterval}}$$

In both cases, we allow all packets which are sent during the last $D_{propagation}$ seconds to get lost.

4.2.4 Key Parameters of the Scenario

In the following table, all parameters described above are defined for the scenario we used for the evaluation. Note, that the parameter `d_suicide_probability` is equal to zero and thus, hosts never fail.

Parameter	Value
`t_join_time_begin` [s]	0
`t_join_time_end` [s]	100
`d_join_probability`	1
`t_leave_time_begin` [s]	600
`t_leave_time_end` [s]	700
`d_leave_probability`	0.5
`t_suicide_time_begin` [s]	100
`t_suicide_time_end` [s]	700
`d_suicide_probability`	0
`t_data_start` [s]	100
`t_data_end` [s]	500
`t_data_interval` [s]	0.1
`t_data_setup` [s]	50
`t_data_propagate` [s]	0.5

4.2.5 Statistical Evaluation

As mentioned, we did only use one topology which was randomly generated. To ensure a statistically sound evaluation of OMCP within this topology, we have run the simulation with the same parameters repeatedly until the mean of all measurements stabilized.

In order to estimate the stability of the values, we derived a simple yet effective procedure. We will explain it for an example metric we call X.
Run the simulation for a few times k, $k \in [1, \infty)$, and then do the following:

1. Calculate the mean of X, $\bar{X}_k$, of all k performed runs.
2. Calculate an error indicator E_X using the difference of $\bar{X}_k$, the mean of all runs, and $\bar{X}_{k-1}$, the mean of all runs except the most recent, as follows:

$$E_X := \frac{|\bar{X}_k - \bar{X}_{k-1}|}{\bar{X}_k}.$$

3. (a) If E_x is less than a suitable maximal value:
   i. Calculate the error indicator for the other metrics until either
      A. The error indicators of all metrics have been calculated: End.
      B. An error indicator is greater than acceptable: Continue with next k runs.
   (b) Else: Continue with next k runs.

The results of the simulation experiments are summarized in the next chapter.

Chapter 5

Results

In the first section, we present the results of the measurements of the three metrics stress, delay and stretch and rate our overlay multicast scheme according to the application profiles specified in the previous chapter. In the second section of this chapter, we will discuss the obstacles we encountered during the evaluation.

5.1 Application Profile Results

The key results of the simulation experiments are given in Table 5.1. In the column labelled “Ideal”, estimations about the ideal values that could be achieved are given. In the next column, the unweighted results of OMCP are shown. In the column labelled “Index”, the quotient of the ideal value and the value of OMCP is given. This is the initial value that is then weighted according to the application profiles. The weighted values are given in the application profile columns. The number of runs we have performed is 400.

We have measured the following three metrics. A more detailed discussion of them can be found in the previous chapter.

Stress This metric measures the efficiency of the overlay multicast protocol. Stress is measured on every physical link and is defined as the average number of identical copies of a data packet that travelled this link during a session. The value given in the table is the average over all physical link. For all links that are used for data delivery, stress is 1 in the optimal case, and zero for all other links. An overlay multicast protocol inherently has a higher overall stress factor if there are nodes which transmit data packets to more than one other host.

Delay This is the average of the end-to-end delay from the source to every receiver. The ideal value is an estimation based on the measured unicast delay.

Stretch This metric measures the overhead incurred by the overlay multicast protocol in terms of latency. It is the average ratio of overlay data delivery delay and unicast delay from the source to every receiver.

Table 5.1: *Results of the simulation experiments.*

Application	Ideal	OMCP	Index	Messaging	Download	Streaming	Conferencing
Metrics							
Stress	1.0	1.5896	0.62909		0.6291	0.5033	0.1887
Delay [ms]	61.886	89.339	0.6927	0.4849			0.3464
Stretch	1.0	1.4330	0.69784	0.2094		0.1396	0.1396
Heuristics							
Simplicity	1.0	0.4	0.4	0.08	0.16	0.12	0.04
Scalability	1.0	0.3	0.3	0.24	0.18	0.21	0.27
Rating	10			5.82	5.42	5.49	5.65

5.1.1 Variation between Runs

The measured values of the means of stress, delay and stretch varied considerably between runs. As explained in the previous chapter, we used an error indicator to decide when we had accumulated enough run results. In Table 5.2, we give an example of how this error indicator varies for different numbers of runs.

Table 5.2: *Error indicator after different numbers of runs*

Runs	1	11	21	31	205
Stress	1	0	0	0	0
Delay	0	0	0	0	0
Stretch	0.18446	0.0059306	0.0038655	0.0018952	0.00053782

5.2 Discussion

The simulation experiments and the evaluation based on the application profiles indicate that our overlay multicast protocol is rather a general-purpose protocol. What can be seen is that it is better suited for applications where delay matters, especially messaging and also conferencing. When raw throughput matters, OMCP is a little bit less suitable.

The stress factor is lower than what we had expected. The average of the maximum of all runs is 5.6315, the average of the mean is 1.5896 and the standard deviation among all physical links is 0.39696 on average. This means that about 66 percent of all links have a stress factor of between 1 and 2.

The average stretch factor of 1.4330 is also quite good. This means that the delay when using the data delivery tree of OMCP is less than 50 percent higher than when a direct unicast connection was used. The maximal stretch value, however, is 5.6315, which reveals that there are hosts which have almost a 500 percent increase of the delay. Fortunately, the standard deviation is only 0.68799 and thus almost 66 percent of all receivers have a stretch factor of 2 or less.

What was said about stretch applies to delay as well because these two metrics are related. The maximal delay was 222.56ms and the average was 89.339ms. With a standard deviation of 35.082, most receivers have a delay below 125ms. Thus, videoconferencing should be feasible with almost all receivers. For comparison we give the unicast delays as well. The average unicast delay is 61.886ms and

the maximum is 89.929ms. This indicates, that video conferencing should be possible with all receivers if a better multicast scheme is used.

What bears repeating is that we did not assume node failures. Some qualitative results of scenarios where this kind of incident is allowed to happen will be discussed in Section 5.2.1.

In addition to the very compact results in Table 5.1, we will present a few plots of a session. The source host has address 0 and we will focus on three of the more interesting subscribers, which are hosts 1, 9 and 10. Data transmission starts at 100 seconds and ends at 700 seconds.

Delay Samples

In OMCP, the unicast delay among all hosts is periodically measured using *Ping* packets. Additionally, we measure the overlay delay from the source to the receivers by timestamping the packets at the source host and evaluating this timestamp at the subscriber. Results of the latter measurements are shown in Figure 5.1 over the course of a session. We eliminated all hosts from the plot for which the delay was constant to make the figure more readable. Only after 275 seconds is the topology stable. An indication that the mesh improvement mechanism of OMCP works is that the delay is monotonically nonincreasing.

Stretch Samples

By taking the quotient of the measured overlay delay and the measured unicast delay, stretch can be calculated easily. This quotient is shown in Figure 5.2 for the same run and the same hosts as the delay samples. These two plots only make sense together. Obviously, the stretch of host 1 increases at around 130 seconds, then goes down twice, while during the same time period, the delay of this host decreases in three steps to about one half of the initial value.

This can be explained by the fact that the stretch factor is calculated as the ratio of the overlay delay and the unicast delay. Both these values are measured quantities and the measurements are subject to queueing delays at six routers on average. Since all unicast delays are measured about every nine seconds, it can happen that the measured value of the unicast delay decreases and thus, stretch increases. Host 10 experiences the same kind of stretch variability later in the session.

Figure 5.1: *Delay samples of the three hosts. All other delays were constant during the session. It can be seen, that it takes about 275 seconds from the time where the first node comes online until the overlay topology becomes stable. This plot shows that no change that incresases the delay occurs and thus the optimization seems to work.*

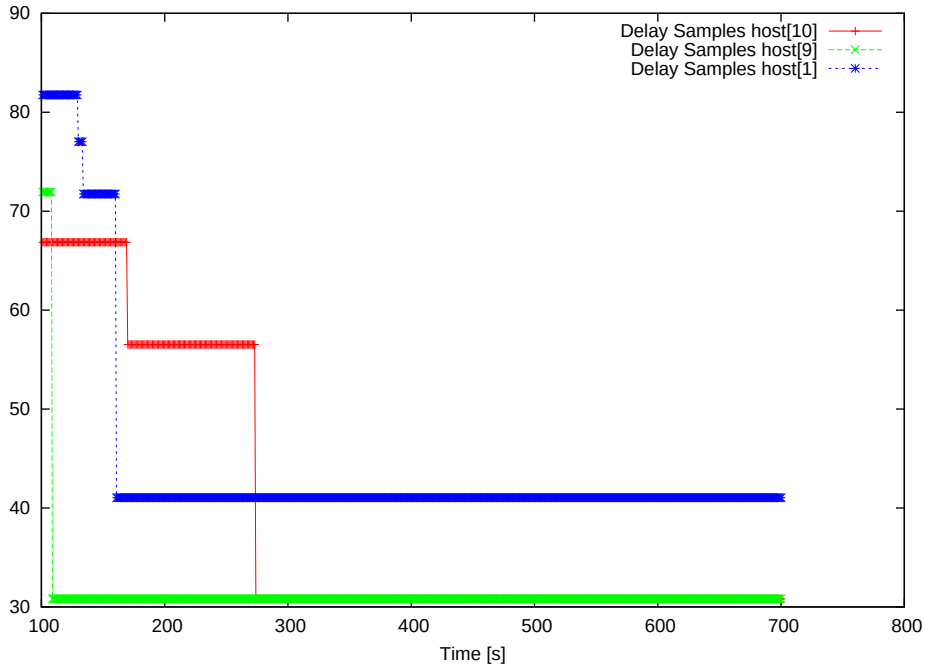
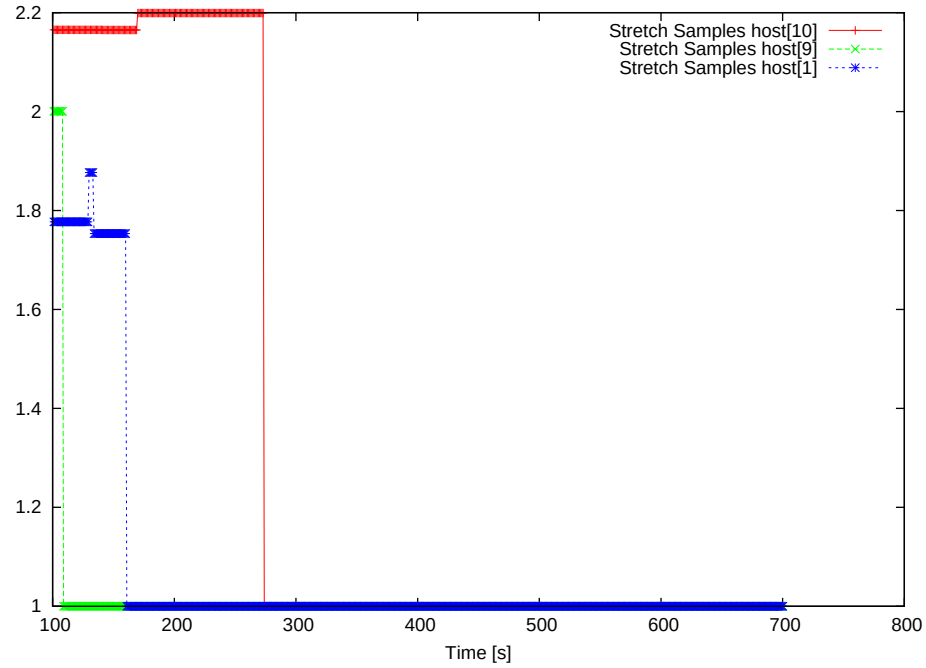


Figure 5.2: *Stretch samples of three hosts. This plot makes only sense when compared to the delay plot above. Interestingly, stretch increases in the curves of hosts 1 and 10, even though the delay decreases at the same time. This is caused by the fact that the unicast delay which as a direct influence on the stretch factor is measured by* Ping *packets and if they are delayed in the queue of a router, the measurements vary.*



Data Delivery Tree

Every host maintains a *parents list* of hosts which he receives data packets from and a *children list* of hosts he sends or forwards data packets to. Figure 5.3 shows the number of children of four hosts. Host 0 is the data source and therefore has the most children.

In Figure 5.4, the current tree parent for the same hosts is shown. (The source is omitted because it does not have a parent for obvious reasons.) It can be seen that all nodes eventually become direct children of the source host 0. At first, host 9 switches its parent from 6 to the source. Thus, it becomes more attractive for hosts 1 and 10. Host 10 also is at first a child of host 6, but after host 9 has become a direct child of the source, host 10 becomes a child of 9. After about 275 seconds, it then determines that it is best to be a direct child of the source and switches to 0. Host 1 first switches from its parent 3 to 6, but about one second later switches to host 9. About ten seconds later, it also notices that it is best to be a direct child of the source and switches to 0.

Figure 5.3: *Number of data delivery tree children of four hosts. All hosts except leaves have children. In this plot, it is indicated, how many children a hosts has over the course of a session. The source, host 0, accumulates the most children, because the algorithm strives to minimize the delay between the source and all receivers. Because the number of receivers is rather small, most of them have one child.*

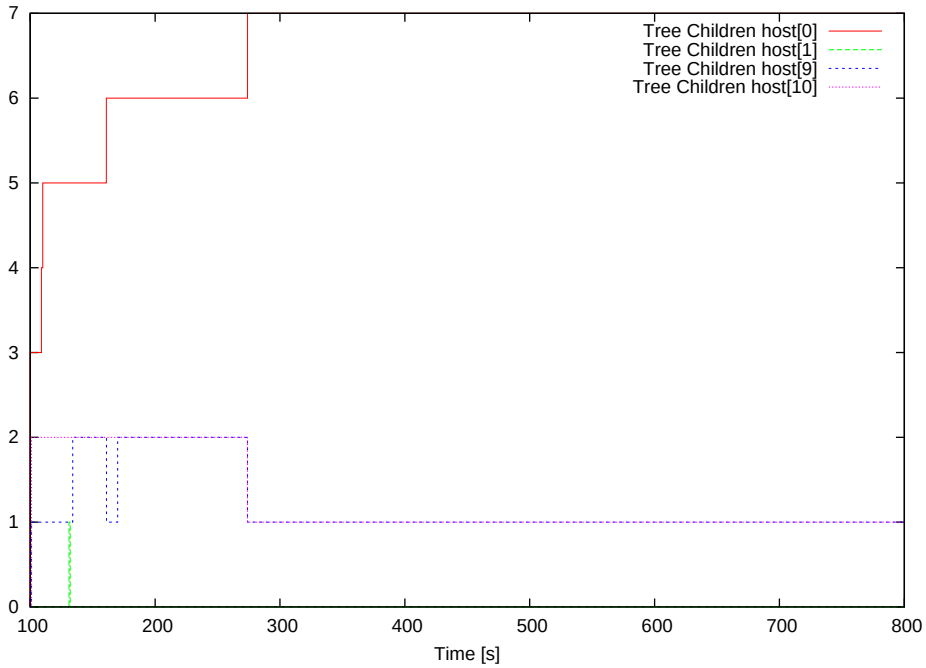
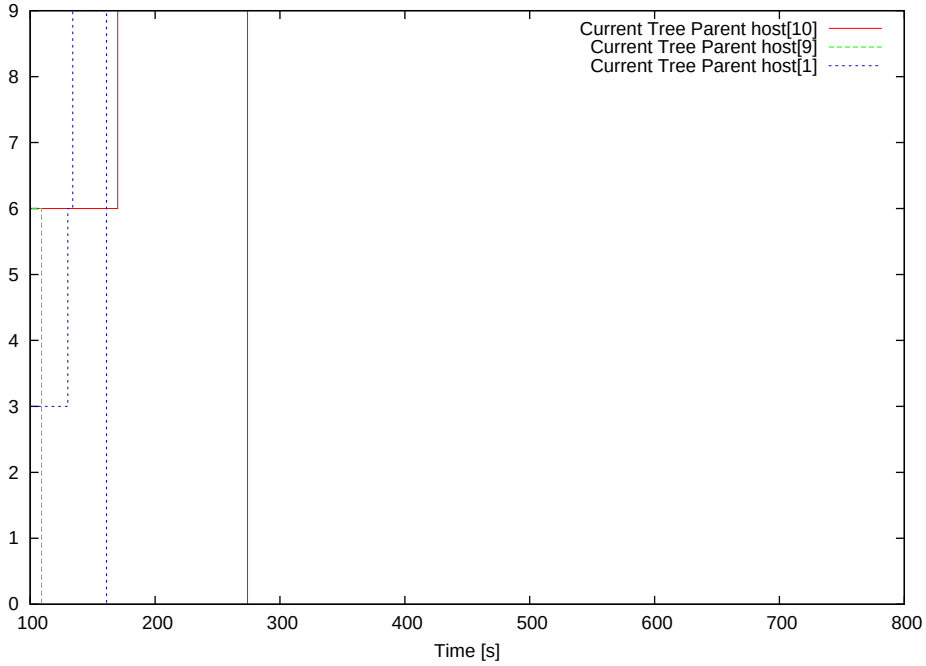


Figure 5.4: *Current parent host of three hosts. (Source is omitted.) All hosts eventually become direct children of the source. First of all, host 9 leaves its parent, host 6, and becomes a child of the source. Then, host 10 determines, that host 9 has a better connection to the source and becomes its child. Host 1 switches from its initial parent, host 3, first to host 6 and then to host 9. But both, hosts 1 and 10, have become direct children of the source after 275 seconds.*



5.2.1 Node failures

In order to provide some insight in the behaviour of OMCP in the case of node failures, we will now present some plots of such a scenario. The parameters are the same as specfified in the previous chapter, except that the paramter `d_suicide_probability` now has the value 0.5. The source host, however, cannot die. Addionally, the `t_data_interval` now has the value 1, which means that the source only emits one packet per second.

In Figure 5.5, the status of all nodes which die during the session is shown. All nodes start in the initial status 2. When they start the joining procedure, their state is 1. Once a host has joined the mesh, its status becomes 0. When it leaves the group, it switches its status to -1, but continues to forward packets for some time. If, however, a node dies abrubtly, its status becomes -2 and it stops processing packets immediately.

It comes as no surprise that node failures lead to packet loss. Since the source host sends one packet per second, the duration of the offline status of a host can directly be determined from the plot in Figure 5.6. Host 6 misses packets twice, first when host 1 dies, and then again when host 4 stops operating. All hosts which are not shown in this plot do not miss any data packets. The packet loss is only detected when a host receives the first packet after a down time. Therefore, the death of a host and the resulting indication of packets loss do not coincide exactly.

Whenever a nodes dies which had children, they perceive packet loss. This happens first to hosts 6 and 10 because their parent, host 9, dies. Later, when node 1 dies, host 6 misses packets again. When nodes 4 and 8 die at about 440 seconds, hosts 5 and, once more, host 6 miss packets.

5.2.2 Obstacles

We were unable to perform all the simulation experiments we had planned due to the very limited timeframe of a semester project. For this reason, we only have results of one specific topology with one particular set of parameters. We suspect that the performance of our scheme could be improved considerably with the appropriate parameters. The parameters we have used in our simulation runs were mere estimations of reasonable initial values.

Missing Information There is currently no freely accessible data about the reliability of Internet backbone routers and links. Internet service providers (ISPs) would be the natural source of such information, but it is just as natural for them to be unwilling to “admit” that their network infrastructure is not infallible.

Topological Limitations Due to time constraints we could only evaluate one network topology. Thus, our results are far from representative. What’s more, the topology we used is much too small to deliver results that can be applied to the Internet.

Figure 5.5: *Status of six hosts which die during the session. Statuses* 2 *and* 1 *indicate that the node has not yet joined or is in the process of joining, respectively. Status* 0 *indicates a fully operational group member. Statuses* -1 *and* -2 *indicate a leaving and a dead host, respectively.*

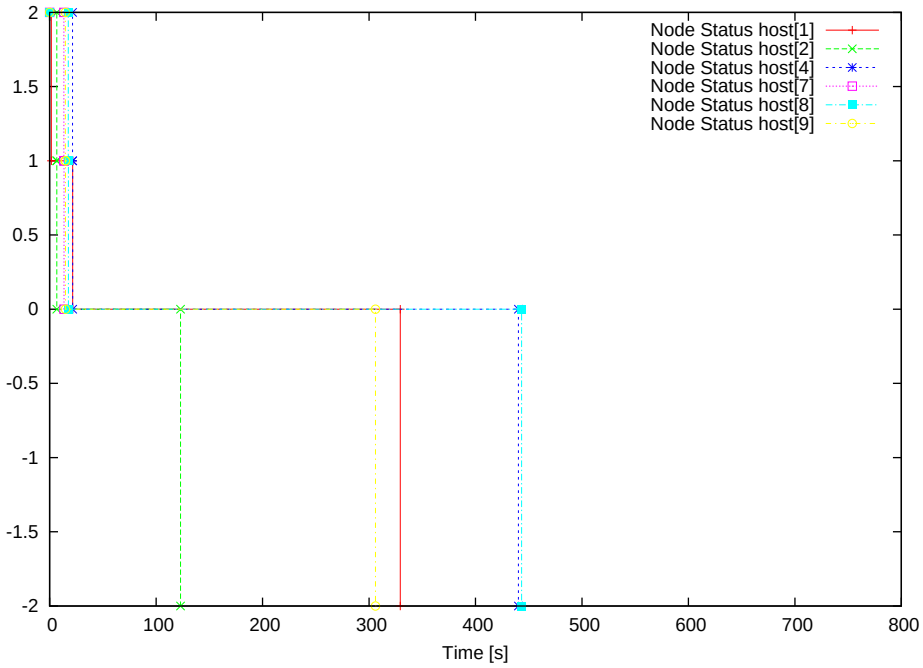
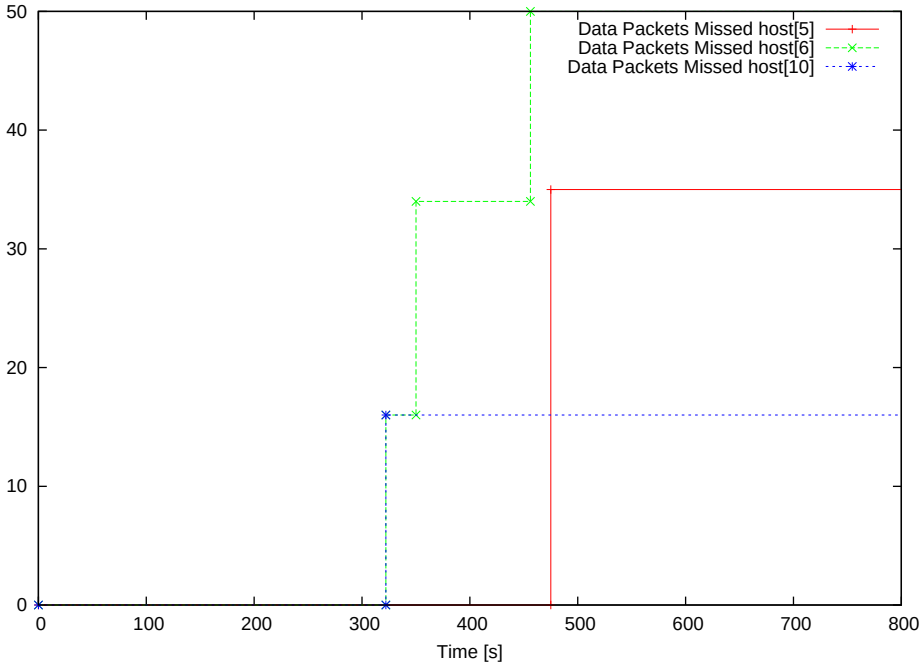


Figure 5.6: *Number of packets missed at three hosts. As indicated by the above plot, four hosts die. When host 9 dies at* $t = 303s$*, both host 6 and 10 miss 16 packets. Caused by the death of node 1, host 6 misses packets again at* $t = 312s$*. At about* $t = 440s$*, when both node 4 and 8 die, hosts 5 and—again—host 6 miss packets.*



Chapter 6

Conclusion

We designed an overlay multicast scheme based on *Narada,* and implemented it in the network simulator *OMNeT*$^{++}$. With this simulator, we have conducted a sufficient number of runs to be able to gather statistically sound values for stretch, delay and stress of one topology and with one set of parameters.

A lot of work could not be done because our time budget was very limited. The behaviour of our scheme under conditions where nodes, routers or links fail, could not be analyzed. However, from a few test runs, it appears that our scheme is *not* suitable for scenarios where node failures are likely because packet loss in this kind of event is too high for most multicast applications, even for fault-tolerant ones like media streaming.

6.1 Further Research

One of the key problems of our scheme is its missing resilience against node failures. As a possible counter measure, we suggest multi-path routing. Additionally, it would be necessary to add a mechanism which allows to immediately detect when a nodes loses its parent in the data delivery tree.

Instead of flooding the network with dedicated probe packets to measure the performance of links, ordinary control packets could be used for this purpose. This would allow a more continuous monitoring of the performance of the underlying network at no additional cost.

The current version of our scheme uses a lot of very small control packets. This could be avoided by simply concatenating the payload of these packets into bigger “summary” packets. Furthermore, data packets could be used to deliver control information as well.

With the extensions sketched above, our scheme might be better suited for real world scenarios. It would be very interesting to analyze, how resilient it is against the kinds of failures that occur in the Internet in a large-scale evaluation.

Appendix A

Conceptual Formulation

On pages 52 to 59, the complete conceptual formulation is given. A review of the project goals reached and not reached follows in appendix B on page 60.

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Aufgabenstellung

von der Semesterarbeit

für

Hr. Simon Heimlicher

"Evaluation of Routing Schemes for Group Communication in Packet Switched Networks"

Aufgabenstellung: Kostas Katrinis

Beginn der Arbeit: 04.11.2002

Abgabetermin: 15.02.2003

Betreuung: Kostas Katrinis - Prof. Bernhard Plattner

1. Introduction

The establishment and wide deployment of numerous emerging applications - like video conferencing, computer-supported collaborative work and network games - depend largely on the availability of a dependable and efficient group communication transport service. Focusing on packet based networks and particularly on the Internet Protocol (IP) as the network layer protocol, group communication is supported by *native IP multicast* [1]. The latter defines group specific addressing and is accompanied by membership (IGMP - Internet Group Management Protocol) and routing protocols (e.g. DVMRP - Distance Vector Multicast Routing Protocol [2]). A wide area testbed called "Multicast Backbone" (MBone) was developed by academic and research affiliators to study IP multicast. MBONE is comprising of geographically dispersed multicast "islands" interconnected with dedicated static unicast tunnels.

Despite the efforts of the networking community, IP multicast is still lacking of wide deployment in the Internet. Possible reasons accounting for this are the inefficiency of inter-domain multicast routing, the non-incremental deployment model, the lack of pricing schemes and the lack of a reliable delivery service and congestion control among others [3]. In an effort to alleviate this problematic situation and yet to enable emergent applications, researchers have come up with alternative ways of realizing group communication in non multicast-capable IP networks: *host-based* multicast and *replicator-based* (or *reflector-based*) multicast.[1]

2. Conceptual Formulation

2.1 A Primer on Overlay Multicast

Host-based multicast ([4],[5]) tries to harness the bandwidth and processing capacity of endsystems to construct distribution overlays. Every node of the overlay can be at the same time both a switching and a sink entity, i.e. a node both forwards packets to its neighbours (in the overlay) and also consumes the received packets (e.g. renders a video stream to the user residing at the endsystem). Figure 1 shows an instance of a multicast overlay consisting of 5 nodes and with node A being currently the source of the distributed packets. Distribution of streams takes place in this overlay as following: A is aware that C is his neighbour in the overlay. As such, it forwards all the packets to C and relies on C to populate the distribution to the other nodes. Subsequently C forwards the packets to its two neighbours, namely B and D and the packet distribution continues in the tree until all the leaf nodes have been reached. The weights assigned to each point-to-point between every two nodes correrponds to the delay between the nodes it is connecting. Note that a link in this figure is an end-to-end link between two endsystems and comprises normally of a path of several physical links. To elaborate more on this, we present in Figure 2 the same overlay in network layer granularity. For instance, the end-to-end link between nodes C and D consisting of a path of three physical links (C-R_3, R_3-R_4, R_4-D) is presented at the network layer. Note that the involved routers are not aware of a multicast transport service, they just provide for unicast connections between the participating end systems. On the contrary, in the native multicast case the routers form a multicast tree that connects all the nodes among the communicating group (Figure 3). Switching is in this scheme performed only by routers and multicast transport is completely transparent to the end systems.

[1] In the bibliography numerous terms are used, as for instance "application layer multicast" or "overlay multicast". For clarity reasons we use the term "host-based" to refer to schemes, where end systems have also switching functionality and the term "replicator-based" to refer to schemes that use only special purpose machines (replicators) to realize multicast and not end systems. Note that both paradigms shall be categorized under application layer multicast schemes. Li et al. prefer the terminology "fixed nodes" and "dynamic nodes" overlays with a similar semantic.

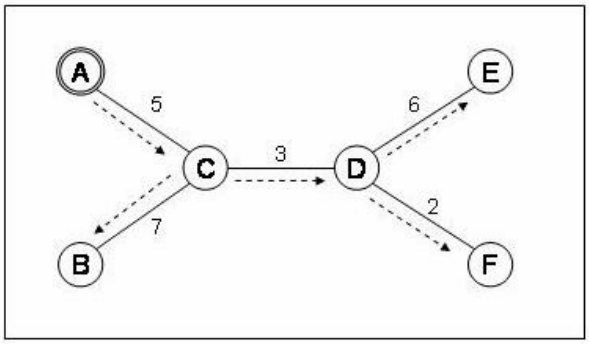


Figure 1 *- Application Layer Overlay*

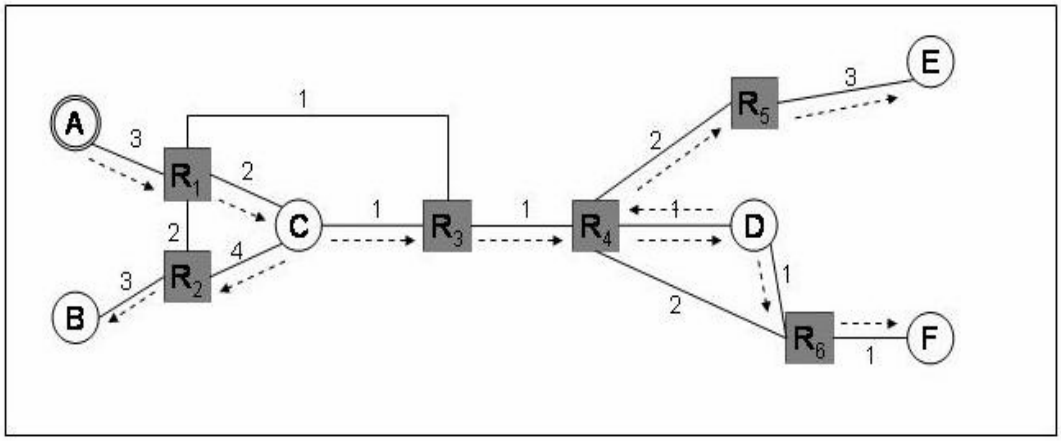


Figure 2 - *The overlay of Figure 1 in network layer granularity*

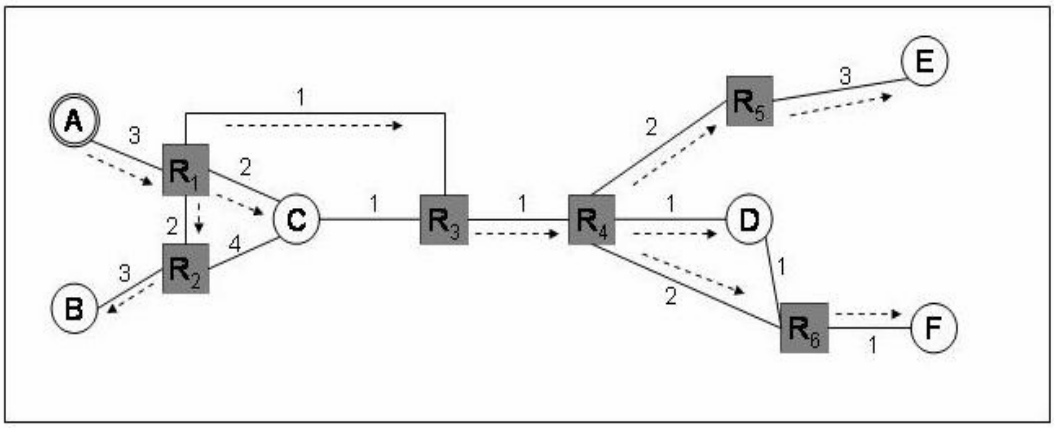


Figure 3 *- Communication among the same node group using native IP multicast transport*

Intuitively, any routing overlay realized on the application layer is not as efficient as a scheme fullfilling the same goals implemented on the network layer. This should become clear if one compares the native multicast scheme presented in Figure 3 with the overlay alternative of Figures 1 and 2 against routing efficiency.

First, it is straightforward that routing delays are higher in the overlay case. For instance, the delay experienced by each packet sent by the source A to node E is 14 in the overlay scheme, whereas IP multicast reduces it to 10 (we even act here optimistically and assume that the endpoints have the same forwarding delays with the routers). Generally, we consider the IP multicast scheme as the optimum and evaluate the delay efficiency of the application layer overlay using a metric called *Relative Delay Penalty* (RDP):

$$|V| \cdot RDP = \Sigma \frac{d_{ov}(v, w)}{d(v, w)} \quad \text{for every node v,w in V}$$

, where $d_{ov}(v,w)$ is the delay between end systems v and w in the overlay (we assume symmetric delays on the links), d(v,w) is the delay between the end systems v and w when IP unicast to connect them is used (lower bound) and V is the set of all end systems (group members) in the group. It is straightforward that RDP will be always greater than 1 and that overlays with RDP values closer to 1 will be more efficient with respect to delay.

Furthermore overlay multicast schemes pose extra load on the network. This lies on the fact that data packets need often cross a single link multiple times. Consider for example in the overlay fn Figure 2 the route of a packet from A to E: the packet first reaches D and then is forwarded by D to E. This has the side-effect that the link between node D and router R_4 is crossed twice by the same data packet. Again the optimum here is native IP multicast, where every data packet crosses every link only once en route to any of the destination end systems. We define a metric called *stress S* to quantify the load that an overlay multicast scheme poses on the network:

$$S = \sum s_e \quad \text{for every link e in E}$$

, where s_e the times that link e is crossed by a single data packet on the overlay and E is the set of links used by the overlay. The stress S is lower bounded by the number |El of links. Obviously one wants to design overlays with low stress.

Similarly to delay overhead and network load, one can define metrics to quantify the efficiency of an overlay with respect to further parameters:

- *end-to-end bandwidth* between every two nodes on the overlay
- *node load bound:* this aspect refers to the maximum number of neighbours that each node is allowed to have in the overlay. Clearly, allowing an unbounded number of neighbours for each node maximizes the topology alternatives, but at the same time it raises fairness issues; nodes with large number of neighbours (*high degree nodes*) would carry much more load than nodes with low number of neighbours (*low degree nodes*).
- *fault tolerance*: an application could for instance require 2-fault-tolerant overlays, i.e. overlays where connectivity is guaranteed at single node failures. For this purpose it is critical to deduce a failure model for such overlays and use it to design overlays that match application requirements.
- *scalability across several dimensions* apart from number of nodes (*group size*), like number of groups and group density (geographical dispersion of nodes). To exemplify this in the case of IP multicast, Distance Vector Multicast Routing Protocol (DVMRP) is not appropriate for sparse groups (large geographical dispersion of nodes) in terms of network resource utilization and network overhead. This lead to the development of multicast routing protocols targeted at sparse groups (PIM-SM).
- *fairness* issues should also be considered, if one wants to achieve a wide deployment of such protocols. For example, many existing approaches measure the average delay between any two nodes in an overlay, but they do not proceed to measure the standard deviation of the delay samples. As such, it is not obvious, whether nodes suffering from considerable delay exist in the overlay. Typical parameters that need to be optimized here are placing load to

nodes proportionally to their capabilities and achieving equal performance gains (like delay from source) on every node in the overlay.

Apart from the quantitative evaluation one can compare the existing scheme categories according to qualitative criteria. Maturity and ease of deployment (incremental for example) are two critical points governing establishment, as well as existence of pricing models. Furthermore specific applications require provision of vertical services to a group communication scheme, like reliable transport and congestion control. Last but not least maintaining secure communications would prove also necessary for specif application domains.

Until now we have not touched on *replicator-based* multicast schemes. Likewise to the host-based case, these are overlays realized on the application-layer. However, in addition to ordinary end systems, they deploy special purpose application-layer elements called *replicators*. As the term conveys, a replicator is a forwarding entity: it forwards to all its neighbours the packets it receives from another neighbour. Ultimately, every node has a replicator as its parent and routing is performed among replicators, i.e. transparently to the end systems. An instance of a replicator-based group communication scheme is shown in Figure 4. Of course one could derive hybrid models, where replicators are used for clustering nodes and routing among clusters and host-based multicast is utilised for intra-cluster routing. A similar problematic with host-based multicast appears also in the replicator-based schemes, with slight differences however. For instance, we expect a more stable failure model, due to the fact that replicators have far more larger online time and failure rate compared to ordinary end systems in the host-based scheme. On the other hand, we face the problem of the calculation of the optimal number of replicators and their optimal placement into the network. Existing experimental schemes of this type are Scattercast [6] amd Akamai's EdgeSuite Streaming [7].

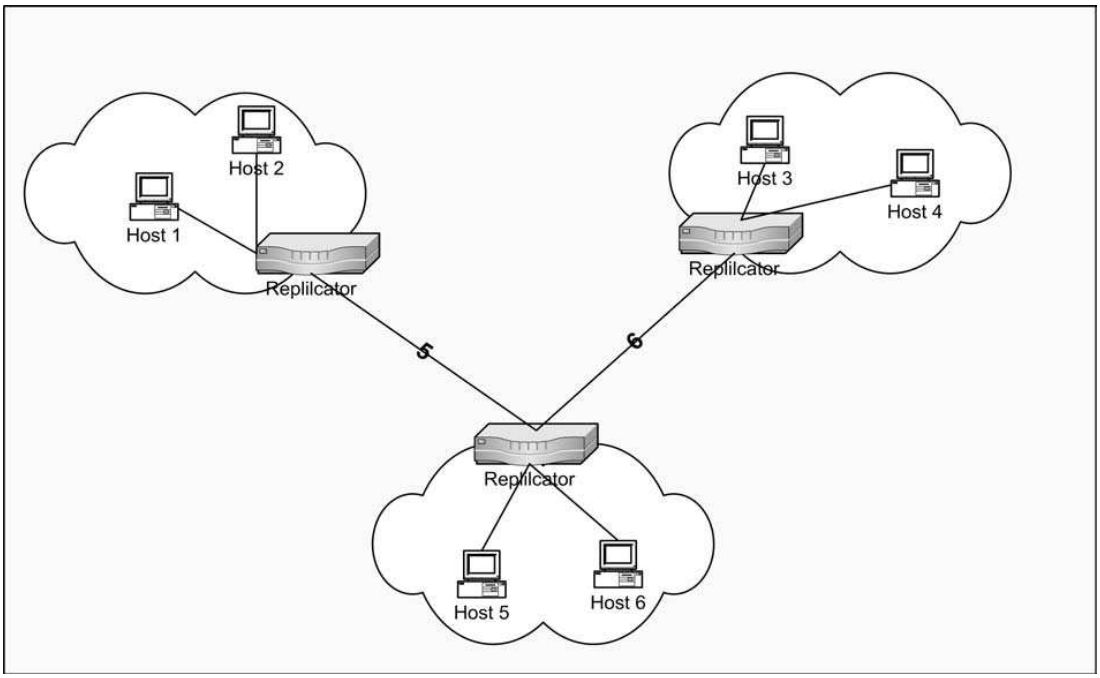


Figure 4 *- Instance of a replicator-based application-layer overlay. In this snaphsot a single replicator serves all hosts of a particular domain, whereas the replicators organize themselves into an overlay according to a given routing protocol.*

2.2 Thesis Objectives

As the title conveys the goal of the present thesis is the evaluation of existing approaches for realizing group communication in the Internet. Departing from the common truth that the Internet is still lacking a globally available multicast service, we arrive at the following question:

" What is the most efficient multicast service for wide deployment in the existing Internet among the existing alternatives?"

We have already shortly reviewed the prominent alternatives, namely IP multicast, host-based- and replicator-based application-layer multicast. The objective is to choose one existing protocol representative for each category and evaluate them against each other according to well defined criteria. It should be noted here that we will not limit the evaluation to comparison of measured primitive performance parameters (like RDP and stress). We will proceed to estimate the suitability of each scheme for specific application domains. To achieve this, we will define first application profiles and then assign weights to every performance parameter for each profile. For instance, a real-time application profile will have a high delay weight and a high network load weight. In this manner we will derive a general score for each scheme and be able to assess the appropriateness of it for a particular applilcation domain. This final step will test the hypothesis that no single multicast scheme is the most efficient for any target application (considering of course accompanying costs).

3. Procedure

Herein we present a draft set of steps towards achieving the thesis objectives:

1. Studying of the literature recommended by the advisor. Please note that the list of references given in the last chapter is not exhaustive for the purpose of this thesis.

2. Selection of the multicast schemes that will be put under study. The selection criteria will be obviously qualitative in this step, however they shall be acceptably justified.

3. Discussion of the evaluation parameters and creation of the application profiles. Here it should also be decided, which parameters will be measured using simulation and which will be qualitatively estimated.

4. Studying of the OMNET++ simulator.

5. Implementation of the chosen protocols in the OMNET++ environment and measurement of the parameters of interest.

6. Processing of the results and derivation of conclusions.

The student may feel free to propose his own approach on the accomplishment of the thesis tasks. *It is highly recommended to work on the documentation in parallel with the treatment of each task.*

4. Important Remarks

- The OMNET++ network simulator [8] shall be used for any simulation-mediated evaluation of the schemes under study. During the working plan creation, the student should not underestimate the learning cycle time of the tool. The selection of the schemes put under study may be conditioned on the existence of available OMNET++ simulation source, as far as they are representative of the multicast paradigm they implement.
- A timeplan for the realization of the semester thesis should be made by the end of the first week and discussed with the supervisor.
- By the end of the thesis you should compose a written report in the form of documentation of the thesis. All abbreviations and not widely established terms must be clearly defined and all assumptions made must be clearly stated.
- The thesis may abstractly be layered into the following seven parts: Introduction, Background Information (including aims and goals), Schemes under study, Evaluation (Methods and Models), Results, Conclusion, References.
- By the end of the second month a short intermediate report should be composed and reviewed during a meeting of the student with the supervisors. The intermediate report should list the already achieved tasks and the tasks that are foreseen to have been accomplished by the end of the thesis. The intermediate report should comply with a structural design of the final report (in a bulleted form).
- An ordinary weekly meeting must shall take place between the student and his supervisor to review the progress of the work and to discuss possible problems and advancement decisions. It is important for the student *to prepare himself for the meeting in advance*. The student may feel free to communicate more frequently with its supervisor via alternative means (e-mail, telephone). Especially vital is the daily reading of e-mails.

5. Thesis Results

A fifteen minutes presentation of the thesis results should be given in the TIK Institute. The exact date of the presentation will be specified late in the semester. Apart from this presentation, the following documents should be handed in on thesis completion:

- A detailed technical report ("Bericht") in english. The following topics should be thoroughly addressed in this report: a description of the investigated research area, a detailed survey of the examined multicast alternatives, a listing of the criteria according to which the schemes have been evaluated and a precise specification of the methods and models used for the evaluation. The results shall be visualized using graphs/tables and conclusions shall be drawn out of them, applied only in the scope of the assumptions made and using only logical reasoning. Finally the report may include a listing of the solved and unsolved problems (together with the reasons why they haven't been solved) and should contain references to literature, table of contents/figures/tables and potential appendices (glossary, programming code, state diagrams, protocol descriptions etc.). The report should end up with an evaluation of how far the initial tasks of the thesis have been achieved and whether the initial timeplan was fulfilled. Four copies of the final report should be handed in, all in paperback form and double sided printed.
- An abstract in both *german and english*, 1-2 pages long. This should contain a quick overview of the performed work. The structure of the abstract should be in the form: (1) Introduction, (2) Aims & Goals, (3) Results, (4) Open Issues.
- An electronic version of the technical report as well as of all the produced documents

(code documentation, models etc.). Figures contained in the final report have to be additionaly stored as independent data in a custom-selected format (ex. EPS). The material in electronic form should be either stored on a CD or in a separate directory on an institute's server (accounts should then be created for the students).

• Referenced and processed literature, whether in electronic or printed form.

• A handbook of the implemented system that should contain: system overview, description of the implementation (structure), documentation on data structures and description of test programms. Moreover installation guidelines and potential hardware/software requirements should be included.

• The complete source code of the system and of the test codes, together with all the necessary libraries/APIs/external programs. Respectively for the system executables and the test programs.

6. Bibliographical References

[1] Wittmann R. , M. Zitterbart, "Multicast Communication: Protocols, Programming, and Applications", Morgan Kaufmann, 2000.

[2] Waitzman D. , C. Partridge and S. Deering, "Distance Vector Multicast Routing Protocol", RFC 1075, Internet Engineering Task Force, 1988.

[3] Almeroth K.C., "The evolution of multicast: from the MBone to interdomain multicast to Internet2 deployment", Network, IEEE, 2000. Vol. 14 Iss. 1: p. 10-20.

[4] Chu Yang-hua, Sanjay G. Rao, and Hui Zhang, "A Case for End System Multicast", ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2000.

[5] Pendarakis D., S. Shi, D. Verma, M. Waldvogel, " ALMI: An Application Level Multicast Infrastructure", 3rd USENIX Symposium on Internet Technologies and Systems, 2000.

[6] Chawathe,Y.D., "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service", PhD Dissertation, University of California at Berkeley, 2000, California.

[7] EdgeSuite Streaming, Akamai, http://www.akamai.com

[8] OMNET++ Discrete Event Simulator, http://www.omnetpp.org

Appendix B

Review

From the initial plan, to implement two overlay multicast schemes and compare them to a native multicast scheme, about one third—the implementation and evaluation of one overlay multicast scheme—was achieved. However, the performance of this custom scheme appears to be far better compared to the original scheme it has been derived from.

The lessons to be learnt in this kind of semester project are:

1. Effectively find and study related work.
2. Develop sound criteria to classify related work and existing solutions and select the most representative of them.
3. Design a procedure to perform a fair, reproducible and comprehensive comparison of a set of solutions.
4. Usage of a network simulator.
5. Implementation of a distributed algorithm, i.e. a multicast protocol running on several nodes.
6. Select parameters of interest and develop the facilities to measure them accurately.
7. Process huge amounts of data and derive statistically sound conclusions if possible.

Even though only one protocol could be implemented and tested, the above—and a lot more—has been learnt thoroughly.

One of the key learnings is the importance of implementation. Or in the words of somebody who obviously has learnt this lesson long ago:

> We reject: kings, presidents, and voting.
> We believe in: rough consensus and running code.
>
> *Dave Clark, IETF*

Appendix C

OMCP Implementation

In this chapter we will first specify all simulation parameters of our overlay multicast scheme OMCP and then present a few excerpts from the source code of the implementation in OMNeT^{++}.

C.1 Simulation Parameters

There are a great number of parameters governing the performance of OMCP. With most of them, there is a trade-off between efficiency and speed as well as resilience against node failures. In general, to make the protocol faster and more adaptable to changes in the scenario, more control traffic is necessary.

The numerical values of the parameters in the following list were determined by hand and are in no way proven to be sound by simulation experiments.

The first character of a parameter name indicates its type.

- `t_...` are time parameters and can be specified either in seconds or with the OMNeT^{++} time notation as follows: `ns` for nanoseconds, `us` for microseconds, `ms` for milliseconds, `s` for seconds, `m` for minutes, `h` for hours and `d` for days.
- `d_...` are double parameters, mostly probabilities.
- `n_...` are integer parameters, mostly numbers of items.

C.1.1 Join, Leave And Death Events

- `t_duration = 1000s`: Duration of one simulation run.
- `double_join_probability = 1.0`: Probability that a node will join the group.
- `t_join_time_begin = 10`,
- `t_join_time_end = 100`: Beginning and end of the interval within which a node that has determined that it will join the group sends its initial join request to the other group members. The exact join time is calculated randomly according to a uniform distribution.
- `double_leave_probability = 0.5`: Probability that a node will leave the group at some point in time.
- `t_leave_time_begin = 600`,

- `t_leave_time_end = 700`: Beginning and end of the interval within which a node that has determined that it will leave the group starts the leave procedure. The exact leave time is calculated randomly according to a uniform distribution.
- `double_suicide_probability = 0.0`: Probability that a node will fail abruptly at some point in time.
- `t_suicide_time_begin = 100`,
- `t_suicide_time_end = 700`: Beginning and end of the interval within which a node that has determined that it will suffer of an abrubt node failure dies. The exact death time is calculated randomly according to a uniform distribution.
- `bool_alive = true`: With this parameter, a node can be defined to be dead from the beginning by setting it equal to `false`.

C.1.2 Bootstrap Process

- `n_bootstrap_nodes_known = 3`: Number of other group members a node knows when bootstrapping the protocol. They are chosen randomly according to a uniform distribution from all valid group member addresses. This set might contain nodes which join only after this node or which are already dead or which will never be alive.
- `n_bootstrap_nodes = 2`: Number of nodes that will actually be contacted in the bootstrap process. They are chosen randomly according to a uniform distribution from the above set of candidates.
- `t_bootstrap_timeout = 10s`: Time to wait for successful connection with the group before restarting the bootstrap process with a new selection from the candidates set.

C.1.3 Refresh Mechanism

- `t_refresh_interval = 500ms`: Interval between dissemination of refresh messages.
- `n_refresh_size = 100`: Maximal number of nodes contained in a refresh message.
- `t_refresh_timeout = 2s`: Time to wait for the next refresh message of a mesh neighbor before sending probe messages to check its status.

C.1.4 Routing Protocol

- `t_routing_latency_initial = 1s`: Initial latency of newly detected routes which have not been evaluated using ping packets yet.
- `t_routing_update_interval = 2s`: Interval between dissemination of routing update messages.
- `t_routing_update_delay = 200ms`: Minimal time between triggered routing updates.
- `t_routing_lifespan = 15s`: Lifespan of routing information.
- `t_routing_poison = 100s`: Poison to add to routes which have expired or which pass notably dead nodes.
- `n_routing_update_size = 50`: Maximal number of route entries in a routing update packet.
- `n_routing_update_path_size = 20`: Maximal length of a path in a routing update packet.

C.1.5 Latency Measurement

- `t_initial_ping_delay = 2s`: Timeframe within which a ping is sent to a newly detected group member. The exact point of time is calculated randomly to avoid flooding this new member with pings.
- `t_ping_interval = 5s`: Interval between evaluating a link.
- `t_ping_timeout = 1s`: Timeout of a ping packet.

C.1.6 Data Source Characteristics

- `adr_t_data_source = 0`: Address of the data source.
- `t_data_start = 100s`: Time at which the data source starts to send data packets.
- `t_data_interval = 10ms`: Interval between data packets.
- `n_data_max = 1000000`: Maximal number of data packets to send.

C.1.7 Mesh Improvement Mechanism

- `n_mesh_neighbors_min = 4`: Absolute minimal number of mesh links a node should have.
- `n_mesh_neighbors_max = 6`: Maximal number of mesh links a node should have.
- `t_mesh_eval_interval = 5s`: Interval between evaluating the mesh.
- `t_mesh_eval_wait = 10s`: Time to wait before re-evaluating the mesh after a link has been added or dropped.
- `t_mesh_link_graceperiod = 10s`: Minimal time to wait before considering dropping a fresh link.
- `t_mesh_link_vacation_timeout = 10s`: Time to wait before potentially dropping a mesh link after the announcment that it should be vacated.
- `t_mesh_link_drop_timeout = 60s`: Time to wait before re-enabling a mesh link after the announcement that it should be vacated.
- `d_mesh_eval_target = 0.5`: Target area in the range between minimal and maximal number of mesh links.
- `d_mesh_eval_utility_min = 0.5`: Minimal utility of a link to be added.
- `d_mesh_eval_cost_max = 5.0`: Maximal cost of a link to be dropped.
- `d_mesh_eval_current_latency_min = 0.002`: Minimal latency of a link to be dropped.
- `d_mesh_eval_current_latency_divider = 10`: Divider to calculate the minimal latency of a link to be dropped.
- `t_member_info_lifespan = 10s`: Lifespan of information about group members which is relevant for dropping links.

C.1.8 Mesh Repair Mechanism

- `t_repair_interval = 5s`: Interval between running the mesh repair process.
- `t_stale_max = 30s`: Time to wait before a stale nodes becomes a zombie node.
- `t_zombie_timeout = 60s`: Time to wait before eliminating the member record of a zombie node.
- `t_probe_timeout = 500ms`: Timeout of a probe packet.
- `n_probes = 2`: Number of probes to send to stale nodes.

C.1.9 Tree Transient Data Forwarding Characteristics

- `t_tree_branch_transient_send = 30s`: Time to continue forwarding data packets to former tree children.
- `t_tree_branch_transient_receive = 60s`: Time to accept data packets from a former tree parent.

C.2 Excerpts from the OMCP Source Code

C.2.1 OMCP Node Class

C.2.2 Member Record Class Header File

This is the header file of the member records of OMCP nodes. Every OMCP node keeps such a member record for all currently online group members. It is used to store information about the status of a member, about the overlay link to it and about its routing table. The latter is used to improve the mesh as well as for the construction of the data delivery tree.

```
// File name: MemberRecord.h

/* Models record of a group member. According to the specification
 *    of Narada, the record must contain the address, the last sequence
 *    number seen by this address and the timestamp of the last seq.
 *    number. In OMC, there are a few other things to be kept in a
 *    member record.
 */

#ifndef MEMBERRECORD_DECL
#define MEMBERRECORD_DECL

#include <vector>
#include "csimul.h"
#include "esmdefinitions.h"
#include "routingTable.h"
#include "RefreshTimeout_m.h"
#include "ZombieTimeout_m.h"
#include "ProbeTimeout_m.h"
#include "PingTimeout_m.h"
#include "RoutingRequestTimeout_m.h"
#include "MeshLinkVacationTimeout_m.h"
#include "MeshLinkDropTimeout_m.h"

using namespace std;

class MemberRecord
{
  // member fields
  private:
    int memberAddress;
    int status;

    /* Mesh dynamics
     */
```

```
    // Status of this mesh link
    int meshLinkStatus;
    MeshLinkVacationTimeout* meshLinkVacationTimeoutPtr;
    MeshLinkDropTimeout* meshLinkDropTimeoutPtr;
    // Timestamp when adding a mesh link to this node
    // was last evaluated
    simtime_t meshLinkEvaluatedTimestamp;
    // Timestamp when the measured latency was last updated
    simtime_t latencyTimestamp;
    // Timestamp when this node was added to the mesh.
    // If zero, then there currently exists no link to it
    simtime_t meshLinkAddedTimestamp;
    // Timestamp when this node was dropped from the mesh
    // If zero, then there never was a link.
    simtime_t meshLinkDroppedTimestamp;

    // Timestamp when this node was added to the tree as a child.
    // If zero, then there currently exists no link to it
    simtime_t treeChildAddedTimestamp;
    // Timestamp when this node was dropped from the tree.
    // If zero, then there never was a link.
    simtime_t treeChildDroppedTimestamp;

    // Timestamp when this node was added to the tree as a parent.
    // If zero, then there currently exists no link to it
    simtime_t treeParentAddedTimestamp;
    // Timestamp when this node was dropped from the tree.
    // If zero, then there never was a link.
    simtime_t treeParentDroppedTimestamp;

    double linkUtility;
    int linkCost;

    /* Tree
     */
    // Timestamp when this node was added to the tree.
    // If zero, then there currently exists no branch to it
    simtime_t treeBranchAddedTimestamp;
    // Timestamp when this node was dropped from the mesh
    // If zero, then there never was a branch.
    simtime_t treeBranchDroppedTimestamp;

    /* Refresh
     */
    // Sequence number sent in last refresh message
    int lastRefreshSeqNumber;

    RefreshTimeout* refreshTimeoutPtr;

    /* Routing
     */
    // Pointer to track RoutingRequestTimeout message
    RoutingRequestTimeout* routingRequestTimeoutPtr;
    // Member's routing table from RoutingUpdate messages
    // Used for construction of the data delivery tree
    RoutingTable routingTable;
```

```
    // Sequence number sent in last routing update message
    int lastRoutingSeqNumber;
    // Timestamp when routing information of this node was last updated
    simtime_t routingTimestamp;

    /* Probing
     */
    // Sequence number *sent* in last probe reply to this member
100 int lastSentProbeSeqNumber;
    // Sequence number *received* in last probe reply from this member
    int lastReceivedProbeSeqNumber;
    // Pointer to track ProbeTimeout message
    ProbeTimeout* probeTimeoutPtr;
    // Counter to track how many probes have been sent to this member
    int probeCount;
    // Timestamp when last probe to this node was sent
    simtime_t probeTimestamp;

110 ZombieTimeout* zombieTimeoutPtr;

    /* Pinging
     */
    // Pointer to track PingTimeout message
    PingTimeout* pingTimeoutPtr;
    // Sequence number *sent* in last ping reply to this member
    int lastSentPingSeqNumber;
    // Sequence number *received* in last ping reply from this member
    int lastReceivedPingSeqNumber;
120 // Timestamp when last ping to this node was sent
    simtime_t pingTimestamp;
    // Latency local node to this member (from probe cycle)
    simtime_t latency;

    // Member's routing table from PingReply messages
    // Used for evaluation of utility of adding a mesh link
    // and cost of dropping a mesh link
    RoutingTable meshRoutingTable;

130 // Refresh cycle timestamp
    simtime_t refreshTimestamp;

  public:

    // constructor
    MemberRecord(int,int,simtime_t,simtime_t);

    // destructor
    ~MemberRecord();
140
    // operator =
    void operator=(const MemberRecord &);

    /* access methods
    */

    int getAddress() const;
```

```
    int  getStatus () const;
    void  setStatus ( int );

    int  getMeshLinkStatus() const;
    void setMeshLinkStatus( int );

    MeshLinkVacationTimeout* getMeshLinkVacationTimeoutPtr() const;
    void setMeshLinkVacationTimeoutPtr(MeshLinkVacationTimeout*);
    MeshLinkDropTimeout* getMeshLinkDropTimeoutPtr() const;
    void setMeshLinkDropTimeoutPtr(MeshLinkDropTimeout*);

    simtime_t  getMeshLinkEvaluatedTimestamp() const;
    void setMeshLinkEvaluatedTimestamp(simtime_t);

    simtime_t  getMeshLinkAddedTimestamp() const;
    void setMeshLinkAddedTimestamp(simtime_t);

    simtime_t  getMeshLinkDroppedTimestamp() const;
    void setMeshLinkDroppedTimestamp(simtime_t);

    simtime_t  getTreeChildAddedTimestamp() const;
    void setTreeChildAddedTimestamp(simtime_t);

    simtime_t  getTreeChildDroppedTimestamp() const;
    void setTreeChildDroppedTimestamp(simtime_t);

    simtime_t  getTreeParentAddedTimestamp() const;
    void setTreeParentAddedTimestamp(simtime_t);

    simtime_t  getTreeParentDroppedTimestamp() const;
    void setTreeParentDroppedTimestamp(simtime_t);

    int  getRefreshSeqNum() const;
    void setRefreshSeqNum(int);

    int  getRoutingSeqNum() const;
    void setRoutingSeqNum(int);

    int  getSentPingSeqNum() const;
    int  getNextSentPingSeqNum();
    void setSentPingSeqNum(int);

    int  getReceivedPingSeqNum() const;
    int  getNextReceivedPingSeqNum();
    void setReceivedPingSeqNum(int);

    int  getSentProbeSeqNum() const;
    int  getNextSentProbeSeqNum();
    void setSentProbeSeqNum(int);

    int  getReceivedProbeSeqNum() const;
    int  getNextReceivedProbeSeqNum();
    void setReceivedProbeSeqNum(int);

    simtime_t  getRefreshTimestamp()  const;
```

```
    void setRefreshTimestamp(simtime_t);

    adr_t getNextHop() const;
    void setNextHop(adr_t);

    int getNumberOfMeshRoutes() const;
    int getNumberOfMeshRoutesTo(adr_t);
    RouteEntry* getMeshRoute(int) const;
    RouteEntry* getMeshRouteTo(adr_t) const;
    adr_t getMeshNextHop(adr_t);
    adr_t getMeshHop(adr_t, int);
    simtime_t getMeshLatency(adr_t);
    void addMeshRoute(adr_t, adr_t, simtime_t, simtime_t);
    void addMeshRoute(adr_t dest, adr_t next,
        simtime_t lat, simtime_t, vector<adr_t>:: iterator path,
        int length);
    void addMeshRoute(RouteEntry*);
    void setMeshRoute(adr_t, adr_t, simtime_t, simtime_t);
    void clearMeshRoutes();

    int getNumberOfMemberRoutes() const;
    int getNumberOfMemberRoutesTo(adr_t);
    RouteEntry* getMemberRouteEntry(int) const;
    RouteEntry* lookupMemberRouteEntry(adr_t);
    adr_t getMemberNextHop(adr_t);
    adr_t getMemberHop(adr_t, int);
    simtime_t getMemberLatency(adr_t);
    void addMemberRoute(adr_t, adr_t, simtime_t, simtime_t);
    void addMemberRoute(adr_t dest, adr_t next,
        simtime_t lat, simtime_t, vector<adr_t>:: iterator path,
        int length);
    void addMemberRoute(RouteEntry*);
    void setMemberRoute(adr_t, adr_t, simtime_t, simtime_t);
    void clearMemberRoutes();

    ZombieTimeout* getZombieTimeoutPtr() const;
    void setZombieTimeoutPtr(ZombieTimeout*);

    RefreshTimeout* getRefreshTimeoutPtr() const;
    void setRefreshTimeoutPtr(RefreshTimeout*);

    ProbeTimeout* getProbeTimeoutPtr() const;
    void setProbeTimeoutPtr(ProbeTimeout*);

    PingTimeout* getPingTimeoutPtr() const;
    void setPingTimeoutPtr(PingTimeout*);

    RoutingRequestTimeout* getRoutingRequestTimeoutPtr() const;
    void setRoutingRequestTimeoutPtr(RoutingRequestTimeout*);

    int getProbeCount() const;
    void setProbeCount(int);
    void incrementProbeCount(int);
    void incrementProbeCount();

    simtime_t getProbeTimestamp() const;
```

```
260     void setProbeTimestamp(simtime_t);

        simtime_t getPingTimestamp() const;
        void setPingTimestamp(simtime_t);

        simtime_t getRoutingTimestamp() const;
        void setRoutingTimestamp(simtime_t);

        simtime_t getLatencyTimestamp() const;
        void setLatencyTimestamp(simtime_t);
270
        simtime_t getLatency() const;
        void setLatency(simtime_t);

        double getLinkUtility () const;
        void  setLinkUtility (double);

        int getLinkCost() const;
        void setLinkCost(int);

280     // CAUTION:
        // This enum has to be kept in sync with the enum memberStatus
        // of class ESMNode in file esmnode.h
        enum memberStatus {freshMember,staleMember,probedMember,
          zombieMember};
        // CAUTION:
        // This enum has to be kept in sync with the enum meshLinkStatus
        // of class ESMNode in file esmnode.h
        enum meshLinkStatus {unusedLink, activeLink , forsakenLink};

290

};

#endif
```

OMCP Node Routing Update Handler Method

This is one of the about 100 methods of the OMCP node class. It handles RoutingUpdate packets which contain the conplete routing table of another group member. OMCP uses path information in order to avoid routing loops. Additionally, routes may be poisoned to indicate that they are either scheduled to be torn down or they are already considered to be down.

```
// File name: OMCNode.cc

// [...]

/* Handles received routing update messages
 * 1. Check, if a member record for the sender exists and if a
 *    mesh link to the sender is active. If not, ignore the packet.
 * 2. Check, if the sequence number in the packet is greater than
 *    the routing sequence number stored in the member record. If not,
10  *    ignore the packet.
 * 3. Update the routingTimestamp in the member record of the sender.
 * 4. Process route after route in the packet as follows:
```

```
 * 4.1. Put the whole routing table contained in the packet into our
 *      member record of the sender.
 * 4.2. Check, if the advertised path contains our address. If yes,
 *      ignore the entry.
 * 4.3. Calcualte the effective latency to the destination :
 *      newLatency = (latency advertised by sender)+(latency to sender)
 *      - If the sender is -- according to the current routing
 *        table -- our next hop to the destination :
 *        Replace our current route with the route from the packet
 *      - If the sender is not currently our next hop, but newLatency
 *        is less than our current latency to the destination : Replace
 *        our current route with the route from the packet
 *      - Else ignore the route.
 */

void ESMNode::routingUpdateHandler(RoutingUpdate* ru)
{
  adr_t senderAddr = ru->getSrc();

  // Get the member record
  MemberRecord* sender = locateGroupMemberRecord(senderAddr);
  // Check if the pointer appears reasonable
  if (sender == NULL) {
    return;
  }
  else {
    int senderStatus = sender->getStatus();
    int senderLinkStatus = sender->getMeshLinkStatus();
    if (senderStatus != freshMember){
      return;
    }
    if (senderLinkStatus == forsakenLink){
      return;
    }
    bool fromMeshNeighbour = false;
    if (meshNeighbourExists(senderAddr)) {
      // Sender is mesh neighbor.
      fromMeshNeighbour = true;
    }
    else {
      // Sender is not mesh neighbor and probably
      // wants to evaluate adding a mesh link to us
    }

    // Check if the sequence number is greater than the last
    int seqNum = static_cast<int>(ru->getSeqNum());
    if (seqNum > sender->getRoutingSeqNum()) {
      // Update routing sequence number
      sender->setRoutingSeqNum(seqNum);

      /* Process routing update
       */

      // Current routing table in the member record of the sender

      // Clear the routing table in the member record
```

```
        // We will add all routes anew as we process the
70      // routing update
        sender->clearMemberRoutes();

        // Get the length of the paths to be able to demux them
        int maxPathLength = static_cast<int>(ru->getPathLength());

        // Get the latency from local node to sender of this
        // RoutingUpdate.
        // We look in the MemberRecord for this because we measure the
        // latency with pings and then write the latency into the
80      // MemberRecord in method "pingReplyHandler()"
        simtime_t senderLatency = sender->getLatency();
        if (senderLatency < 0) {
          // Something went wrong. Should never happen.
          senderLatency =  t_routing_latency_initial_M ;
        }

        // Number of entries in this RoutingUpdate
        unsigned int numEntries = ru->getNumEntries();

90      /* Check if the mesh link to the sender is active.
         * If its status is forsakenLink, then check if it uses
         * us as next hop for any route.
         * If yes, then we need to reschedule the VacationTimeout
         * because we wait until all routes have left this link.
         */
        bool forsakenLinkStillUsed = false;

        /* Check if we received one or more poisoned routes we
         * believed were good. If yes, we send out routing updates
100     * in at most t_routing_update_delay_M seconds.
         */
        bool sendUpdatesImmediately = false;

        // Go through all vectors in RoutingUpdate
        // simultaneously with index i
        for (unsigned int i = 0; i < numEntries; i++){
          // Get the destination address of this route
          adr_t destAddr = static_cast<adr_t>(ru->getDestination(i));

110       /*
           * RoutingUpdate conventions:
           * - The very first entry is always of the form
           *   Destination = <senderAddress> NextHop = <senderAddress>
           *   Latency = 0;
           * - The following entries have different destinations than
           *   <senderAddress>.
           * - The end of valid entries is reached if an entry to
           *   destination <senderAddress> appears.
           */
120
          if (i > 0 && destAddr == senderAddr) {
            // This route must be invalid. Quit processing this packet
            break;
          }
```

```
else {

  /* Valid route
   */

  // Update the routing information timestamp
  // of the sender
  sender->setRoutingTimestamp(simTime());

  adr_t nextHop = static_cast<adr_t>(ru->getNextHop(i));
  simtime_t lat = static_cast<simtime_t>(ru->getLatency(i));

  /* Copy the path into a vector and check if it contains
   * our address. If yes, ignore the entire entry and go
   * to the next.
   */
  bool pathContainsUs = false;
  // Get the beginning of the path in the array
  int base = maxPathLength * i;
  // Set the index in the path for this record to the first hop
  int pathIndex = base;
  // Count the number of valid hops in the path
  int pathLength = 0;
  vector<adr_t> pathVector;
  pathVector.clear();
  // Set the first hop in the path to
  // the sender of this packet
  pathVector.push_back(senderAddr);
  pathLength++;
  for (int hop = 0; hop < maxPathLength; hop++) {
    pathIndex = base + hop;
    adr_t hopAddr = ru->getPath(pathIndex);
    // Check if this is a valid hop address
    if (hopAddr == this->localAddress_M){
      // Set variable to not incorporate this route into our
      // routing table. But we still need the whole path
      // to put it in the member record of the sender.
      pathContainsUs = true;
      pathVector.push_back(hopAddr);
      pathLength++;
    }
    else if (hopAddr != ROUTING_NO_HOP){
      pathVector.push_back(hopAddr);
      pathLength++;
    }
    else {
      break;
    }
  }

  /* Update routing tables
   *
   * - If this RoutingUpdate was sent by a mesh neigbhbour:
   *   Add the routes to our routing table if they seem useful.
   * - Else only add them to the MemberRecord of the sender.
   */
```

```
        /* Copy routing information into the member record of the
         * sender
         * At the beginning, we cleared the routing table. Therefore,
         * we can now just create a new entry for all routes found.
         * We won't add routes which are poisoned, however.
         */

        // Check, if this route has been poisoned, i.e., its
190     // latency is equal to or greater than t_routing_poison_M
        bool poisonedRoute = false;
        if (!(lat < t_routing_poison_M)){
          poisonedRoute = true;
        }
        if (!poisonedRoute){
          vector<adr_t>::iterator pathIter = pathVector.begin();
          /* The first entry in the path vector is the address of
           * the sender. We only need this in our own routing table.
           * Therefore, we increment the pathIter once here.
200        * Example: path vector is "3 2 1".
           * - Our routing table will contain:
           * dest: 1, next: 3, path length: 3, path: 3 2 1
           * - The member routing table of sender 3 will contain only:
           * dest: 1, next: 2, path length: 2, path: 2 1
           */
          pathIter++;
          sender->addMemberRoute(destAddr, nextHop, lat, simTime(),
                pathIter, pathLength - 1);
        }
210
        if (fromMeshNeighbour){

          /* Update our routing table
           */

          // Ignore routes to us
          if (destAddr == this->localAddress_M){
            continue;
          }
220
          // Check, if we already have an entry for this destination
          RouteEntry* re = routes_M.lookupEntry(destAddr);
          if (re == NULL){

            /* New destination
             */

            // If the path contains us, ignore the route.
            if (pathContainsUs){
230           if (senderLinkStatus == forsakenLink){
                // The mesh link to the sender should be vacated, but
                // this member is still using us in this route.
                // We need to wait some more time.
                forsakenLinkStillUsed = true;
              }
              continue;
```

```
    }

    // We won't add a new route if it is poisoned.
    simtime_t effectiveLatency = lat + senderLatency;
    if (! poisonedRoute){

      // Create new route entry in the table
      vector<adr_t>:: iterator pathIter = pathVector.begin();
      routes_M.addRoute(destAddr, senderAddr,
          effectiveLatency , simTime(), pathIter , pathLength);
    }
  }
  else {

    /* Known destination
     */
    // Calculate the effective latency if we took this route
    // with the sender as next hop
    simtime_t effectiveLatency = lat + senderLatency;
    simtime_t currentLatency = re->getLatency();
    bool updateRoute = false;
    // Update the entry if the sender is currently our next
    // hop because in this case, the sender is authoritative
    // about link failures etc.
    adr_t currentNextHop = re->getNextHop();
    if (senderAddr == currentNextHop){
      // If the path contains us, we need to poison
      // this route to avoid a routing loop.
      if (pathContainsUs){
        routes_M.poisonRouteTo(destAddr, t_routing_poison_M);
        continue;
      }
      else if (poisonedRoute) {
        // If the route is poisoned:
        // 1. Check if we already have poisoned the route in
        //    our routing table. If not we set the latency to
        //    our own value of a poisoned route and send
        //    routing updates to our neighbors immediatley,
        //    i.e. set sendUpdatesImmediatley to true
        if ( ceil (currentLatency) == ceil (t_routing_poison_M)){
          // We already know that this route is poisoned.
        }
        else {
          routes_M.poisonRouteTo(destAddr,
              t_routing_poison_M);
          // Set this variable to true to make sure we send
          // this information to our neighbors immediately.
          sendUpdatesImmediately = true;
          continue;
        }
      }
      else {
        updateRoute = true;
        // Check if our current route in the routing table is
        // poisoned.
        // If yes, we send routing updates to our neighbors
```

```
        // immediatley, i.e. set sendUpdatesImmediatley to
        // true
        if (ceil(currentLatency) == ceil(t_routing_poison_M)){
          // Set this variable to true to make sure we send
          // this information to our neighbors immediately.
          sendUpdatesImmediately = true;
        }
      }
    }
    // Poison the entry if the old entry is too old
    else if (re->getTimestamp() < simTime() - t_routing_lifespan_M) {
      // If the path contains us, go to the next entry.
      if (pathContainsUs){
        routes_M.poisonRouteTo(destAddr, t_routing_poison_M);
        continue;
      }
      else if (poisonedRoute) {
        // If the advertised route is poisoned, we set
        // the latency to our own value of a poisoned route
        routes_M.poisonRouteTo(destAddr, t_routing_poison_M);
        continue;
      }
      else {
        // This route seems to be valid and our current route
        // has expired.
        updateRoute = true;


        // Check if our current route in the routing table is
        // poisoned.
        // If yes, we send routing updates to our neighbors
        // immediatley, i.e. set sendUpdatesImmediatley to
        // true
        if (ceil(currentLatency) == ceil(t_routing_poison_M)){
          // Set this variable to true to make sure we send
          // this information to our neighbors immediately.
          sendUpdatesImmediately = true;
        }
      }
    }
    // Change the entry if the new route is better
    else if (effectiveLatency < currentLatency) {
      // If the path contains us, go to the next entry.
      if (pathContainsUs){
        continue;
      }
      else {
        updateRoute = true;
        // Check if our current route in the routing table is
        // poisoned.
        // If yes, we send routing updates to our neighbors
        // immediatley, i.e. set sendUpdatesImmediatley to
        // true
        if (ceil(currentLatency) == ceil(t_routing_poison_M)){
          // Set this variable to true to make sure we send
          // this information to our neighbors immediately.
```

```
                    sendUpdatesImmediately = true;
                  }
                }
              }
              // Do the update if necessary
              if (updateRoute){
                re->setNextHop(senderAddr);
                re->setLatency( effectiveLatency );
                re->setTimestamp(simTime());
                vector<adr_t>:: iterator pathIter = pathVector.begin();
                re->setPath( pathIter , pathLength);
              }
            }
          }
        }
        if ( forsakenLinkStillUsed ){
          // Reschedule VacationTimeout
          MeshLinkVacationTimeout* vacationTimeout
              = sender->getMeshLinkVacationTimeoutPtr();
          if (vacationTimeout != NULL){
            // Seems to be a valid timeout pointer
            delete cancelEvent(vacationTimeout);
          }
          sender->setMeshLinkVacationTimeoutPtr(NULL);
          // Schedule a new timer to check if the link has been vacated
          // already
          vacationTimeout
            = new MeshLinkVacationTimeout(
                "Mesh Link Vacation Timeout");
          vacationTimeout->setKind(MeshLinkVacationTimeoutEvent);
          vacationTimeout->setMemberAddress(senderAddr);
          sender->setMeshLinkVacationTimeoutPtr(vacationTimeout);
          scheduleAt(simTime()+t_mesh_link_vacation_timeout_M,
              vacationTimeout);
        }
      }
      if (sendUpdatesImmediately){
        if (routingUpdateEventMsg->arrivalTime()
            > ceil (simTime() + t_routing_update_delay_M)){
          // Cancel the current timer
          cancelEvent(routingUpdateEventMsg);
          // Schedule triggered update
          routingUpdateEventMsg
            = new cMessage("Triggered Routing Update Timer");
          routingUpdateEventMsg ->setKind(ROUTING_UPDATE_EVENT);
          scheduleAt(simTime()
              + t_routing_update_delay_M, routingUpdateEventMsg);
        }
        else {
        }
      }
    }
  }
}

// [...]
```

Bibliography

[1] Y. Chu, S. G. Rao, S. Seshan, H. Zhang, "A Case for End System Multicast." *Proceedings of the ACM SIGMETRICS*, June 2000.

[2] E. B. Shapiro. "Network Timetable." IETF RFC 4 (Informational), March 1969.

[3] J. Postel. "Internet Protocol." IETF RFC 791 (Standard), September 1981.

[4] D. Waitzman, C. Partridge, S. E. Deering. "Distance Vector Multicast Routing Protocol." IETF RFC 1075 (Experimental), November 1988.

[5] J. Moy. "Multicast Extensions to OSPF." IETF RFC 1584 (Proposed Standard), March 1994.

[6] A. Ballardie, P. Francis, J. Crowcroft. Core Based Trees. *Proceedings of the ACM SIGCOMM*, September 1993.

[7] A. Ballardie. "Core Based Trees (CBT) Multicast Routing Architecture." IETF RFC 2201 (Experimental), September 1997.

[8] A. Ballardie, B. Cain, Z. Zhang. "Core Based Trees (CBT) Multicast Routing Architecture." IETF Internet-draft "draft-ietf-idmr-cbt-spec-v3-01.txt", August 1998.

[9] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. E. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei. "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification." IETF RFC 2362 (Experimental), June 1998.

[10] B Fenner, M. Handley, H. Holbrook, I. Kouvelas. "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification (Revised)." IETF Internet-draft, February 2004.

[11] S. E. Deering, R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification." IETF RFC 2460 (Draft Standard), December 1998.

[12] J. Frankel, T. Pepper. "Gnutella."
`http://rfc-gnutella.sourceforge.net/`

[13] B. Cohen. "BitTorrent."
`http://www.bitconjurer.org/BitTorrent/`

[14] "Internet Engineering Task Force (IETF)."
`http://www.ietf.org/`

[15] András Varga. "OMNeT^{++}– Objective Modular Network Testbed in C^{++}." Discrete event simulation system.
`http://www.omnetpp.org/`

[16] Georgia Institute of Technology/College of Computing. "GT-ITM –Georgia Institute of Technology Internet Topology Model." Network topology generator.
`http://www.cc.gatech.edu/projects/gtitm/`

[17] A. Tanenbaum. *Computer Networks.* Published by Prentice Hall PTR, 4th ediction (August 2002), ISBN: 0130661023.

[18] L. Peterson, B Davie. *Computer Networks: A Systems Approach.* Published by Morgan Kaufmann, 3rd edition (May 2003), ISBN: 155860832X.

[19] J. Postel. “Transmission Control Protocol.” RFC 793 (Standard), September 1981.

[20] L. Kleinrock. “Information Flow in Large Communication Nets.” Ph.D. thesis, Massachusets Institute of Technology, May 1961.

[21] J. D. Day, H. Zimmermann. “The OSI Reference Model.” *Proceedings of the IEEE,* December 1983.

[22] International Organization for Standardization. “Open System Interconnection Reference Model.” ISO/IEC Standard 7498, 1984.

[23] S. E. Deering. “Host Extensions for IP Multicasting.” IETF RFC 1112 (Standard), August 1989.

[24] S. E. Deering, D. R. Cheriton. “Host groups: A multicast extension to the Internet Protocol.” *Proceedings of the ACM/IEEE Data Communications Symposium,* September 1985.

[25] S. E. Deering. “Multicast Routing in Internetworks and Extended LANs.” *Proceedings of the ACM SIGCOMM,* August 1988.

[26] S. E. Deering. “Multicast Routing in a Datagram Internetwork.” Ph.D. thesis, Stanford University, Electrical Engineering Department, December 1991.

[27] H. Erikson. “MBONE: The Multicast Backbone.” *Communications of the ACM,* August 1994.

[28] T. M. Munzner, E. Hoffmann, K. Claffy, B. Fenner. “Visualizing the Global Topology of the MBone.” *Proceedings of the IEEE InfoVis,* October 1996.

[29] Y. Sh. Shi. “Design of Overlay Networks for Internet Multicast.” Ph.D. thesis, Washington University, Sever Institute of Technology, August 2002.

[30] F. Lau, H. Rubin, M. H. Smith, L. Trajovic. “Distributed Denial of Service Attacks.” *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* October 2000.

[31] H. Holbrook, D. Cheriton. “IP Multicast Channels: EXPRESS Support for Large-scale Single Source Applications.” *Proceedings of the ACM SIGCOMM,* September 1999.

[32] H. Holbrook, B. Cain. “Source Specific Multicast.” IETF Internet-draft, March 2000.

[33] J. H. Saltzer, D. P. Reed, D. D. Clark. “End-To-End Arguments in System Design.” *ACM Transactions on Computer Systems,* November 1984.

[34] S. Banerjee, B. Bhattacharjee. “A Comparative Study of Application Layer Multicast Protocols.” University of Maryland, October 2002.

[35] P. Francis. “Yoid: Extending the Multicast Internet Architecture.” White paper, April 2000. `http://www.aciri.org/yoid/`

[36] B. Zhang, S. Jamin, L. Zhang. “Host Multicast: A Framework for Delivering Multicast to End Users.” *Proceedings of the IEEE INFOCOM,* June 2002.

[37] S. Banerjee, B. Bhattacharjee, C. Kommareddy. “Scalable Application Layer Multicast.” *Proceedings of the ACM SIGCOMM,* August 2002.

[38] S. RAtnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. “A Scalable Content-addressable Network.” *Proceedings of the ACM SIGCOMM,* August 2001.

[39] M. Castro, P. Druschel, A-M. Kermarrec, A. Rostron. “SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC),* 2002.

[40] C. L. Hedrick. “Routing Information Protocol.” IETF RFC 1058, June 1988.

[41] G. Malkin. “RIP Version 2.” IETF RFC 2453 (Standard), November 1998.

[42] A. S. Thyagarajan, S. E. Deering. “Hierarchical Distance-Vector Multicast Routing for the MBone.” *Proceedings of the ACM SIGCOMM*, August 1995.

[43] J. Moy. “OSPF specification.” IETF RFC 1131 (Proposed Standard), October 1989.

[44] J. Moy. “OSPF Version 2.” IETF RFC 2328 (Standard), April 1998.

[45] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel. “ALMI: An Application Level Multicast Architecture.” *Proceedings of the USENIX USITS*, March 2001.

[46] S. Banerjee, Ch. Kommareddy, K. Kar, B. Bhattacharjee, S. Khuller “Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications.” *Proceedings of the IEEE INFOCOM*, April 2003.

[47] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, J. W. O’Toole, Jr. “Overcast: Reliable Multicasting with an Overlay Network.” *Proceedings of the USENIX OSDI 4*, October 2000.

[48] Y. D. Chawathe. “Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service.” Ph.D. thesis, University of California at Berkeley, December 2000.

[49] B. Cain, S. E. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan. “Internet Group Management Protocol, Version 3.” IETF RFC 3376 (Proposed Standard), October 2002.

[50] NIST/SEMATECH. “e-Handbook of Statistical Methods.”, July 2004. `http://www.itl.nist.gov/div898/handbook/`

[51] I. Busse, B. Deffner, H. Schulzrinne. “Dynamic QoS Control of Multimedia Applications based on RTP.” *Proceedings of the First International Workshop on High Speed Networks and Open Distributed Platforms*, June 1995.

[52] F. Dressler: “A Metric for Numerical Evaluation of the QoS of an Internet Connection.” *Proceedings of 18th International Teletraffic Congress (ITC18)*, August 2003.

[53] P. Clément, B. Jenkins. “Adapting Test and Measurement Tools to Centralcasting and Broadband IP Contribution.” Thales Broadcast and Multimedia, 2003. `http://www.broadcastpapers.com/testmeasurement/ThalesAdaptTestMeasTools.pdf`

[54] F. Dressler. “How to Measure Reliability and Quality of IP Multicast Services?” *Proceedings of 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM ’01)*, August 2001.

[55] Akamai. “EdgeSuite: A Comprehensive Content Delivery Solution for Advanced E-Business.” `http://www.akamai.com/en/html/services/edgesuite.html`

[56] University of Southern California/Information Sciences Institute. “The Network Simulator – NS-2.” Discrete event simulation system. `http://www.isi.edu/nsnam/ns/`

[57] K. Calvert, M. Doar, E. W. Zegura. “Modeling Internet Topology.” *IEEE Communications Magazine*, June 1997.

Index