LULEÅ TEKNISKA UNIVERSITET

1999:037

MASTER'S THESIS

An Examination of the Scalability of Multicast Interactive Applications

Roland Parviainen

Civilingenjörsprogrammet

Institutionen för Systemteknik
Avdelningen för Programvaruteknik

1999:037 • ISSN: 1402-1617 • ISRN: LTU-EX--99/037--SE

An examination of the scalability of multicast interactive applications

Roland Parviainen
Centre for Distance-spanning Technology
Department of Computer Science,
Luleå University of Technology,
971 87 Luleå, Sweden

Roland.Parviainen@cdt.luth.se

January 25, 1999

Abstract

This Master Thesis discusses some issues of scalability in interactive applications that uses multicast. Two prototype applications made by the author that highlights some problems of scalability are also described.

The first application, mIR, multicast Interactive Radio, is a system for distributing high quality audio that provides interactive services to the listeners. The other application is m3DE, multicast 3D Environment, a distributed multiuser 3D environment, where users can interact with each other.

Conclusions about the applications are that mIR is scalable if the problem of reliability is solved by redundancy and not a reliable multicast protocol. For m3DE, it is the application itself, especially the memory requirements that are the restricting factor for scalability.

Preface

This report describes my master of science thesis, which was done during the summer and fall of 1998 at the Software engineering department of the Centre for Distance-spanning Technology.

Part of this work was done within Esprit project 28410 Roxy, which is supported by the Information technology part of the 4:th Framework Program of the European Union. Support was also provided by the Centre for Distance-spanning Technology (CDT), and the Swedish National Board for Industrial and Technical Development(NUTEK).

I would like to thank my supervisor Dick Schefström. I would also like to thank the rest of the staff of CDT for commenting on the work, especially Peter Parnes, who also had the original ideas about mIR.

Luleå, December 1998
Roland Parviainen

Contents

Chapter 1

Introduction

Some define interactive applications as “A term describing a program whose input and output are interleaved, like a conversation, allowing the user’s input to depend on earlier output from the same run.”[1], i.e. the user interacts only with the program itself. All applications that have a user interface can be called interactive with this rather broad definition. In the applications studied in this thesis the word interactive is used to describe applications where the user can interact with other users, i.e. the interactivity is not between the user and the application, but between the different users.

All applications described in this thesis have been developed in the platform independent programming language Java [8]. The m3DE application was developed completely from scratch, and the mIR applications where based on Peter Parnes[2] initial implementation of mIR. This implementation did not have any interactivity, the only supported bit rate was 128 kbit/s and it did not follow the RTP [13][12][7] specifications completely.

1.1 Goals

The main goal of this thesis was to develop two prototypes of interactive applications and to examine some scalability issues of these applications.

The first application is mIR – multicast Interactive Radio. The goal was to make scalable implementations of as much of the initial ideas for mIR as possible[3].

The main ideas were: the server part of mIR would multicast MPEG audio files on different channels depending on the music type. The client part would act as a tuner where the user can choose which type of music she/he wants to listen by selecting the channel. The tuner would also allow the user to vote for the next song to be played.

For the second application, the multicast 3D environment, the goal was to develop a virtual 3D world that users can interact in. IP multicast should be used as the underlying information transport protocol.

[1] Foldoc, <URL:http://wombat.doc.ic.ac.uk/foldoc/>

[2] <URL:http://www.cdt.luth.se/~peppar/>

[3] <URL:http://www.cdt.luth.se/~peppar/progs/mIR/ideas.html>

1.2 Technical background

1.2.1 IP multicast

IP multicast [2] provides efficient many-to-many data distribution in an internet environment. Senders send datagrams to a "host group", a set of zero or more host identified by a single IP destination address. The datagrams is delivered to all members of the host group by the network infrastructure in an optimized way. Neither receivers nor senders need to know who or where the other receivers and/or senders are. The membership of a host group is dynamic; hosts may join and leave groups at any time.

1.2.2 Real-time Transport Protocol

The Real-time Transport Protocol, RTP [13], is a standard protocol for transmitting real-time data such as audio and video. It was explicitly designed for multicast in mind, and is typically run on top of UDP [11]. However, RTP may be used with other underlying network or transport protocols. RTP provides no additional reliability and assumes that low levels of loss are acceptable for audio and video applications.

The data transport is augmented by a control protocol, the RTP Control Protocol (RTCP), which consists of session messages sent periodically to the same destination as the data.

RTP Packet

The RTP data packet header contains the following among other things:

- A sequence number, that can be used to discover packet loss and out of order packets.
- A synchronization source (SSRC) which identifies the source.
- A timestamp.
- A payload type field identifying the format of the RTP payload.
- A marker bit. The interpretation of the marker bit is defined by a profile. For audio transmission, the marker bit is usually used for signaling the start of a talk-spurt.

RTCP

RTCP is based on periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. RTCP packets can contain, among other things, any of the following:

- Source description items (SDES) to identify the participant (name, email, etc) and associate the information with the SSRC in the RTP packets.
- Sender reports for transmission and reception statistics from participants that are active senders.
- Receiver reports for reception statistics from participants that are not active senders.

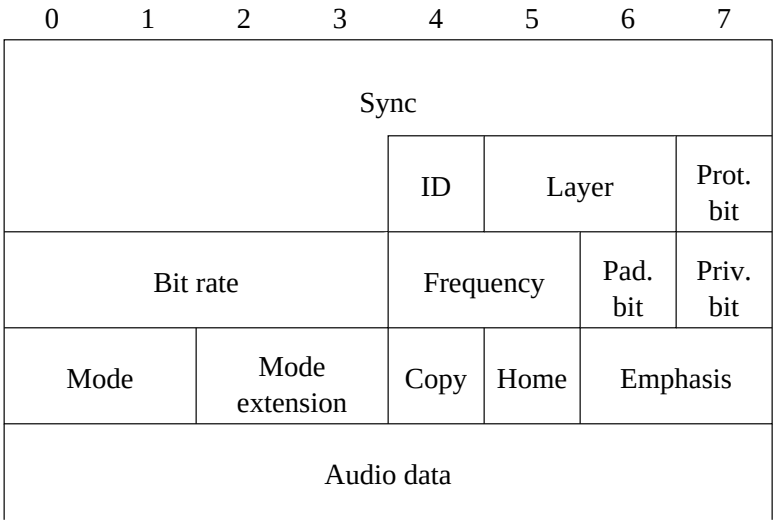

Figure 1.1: The MPEG Audio header

1.2.3 MPEG-1 Audio

MPEG is a working group in a subcommittee of ISO/IEC (the International Standards Organization/International Electrotechnical Commission) that generates generic standards for digital video and audio compression.

MPEG works in phases. These phases are denoted by MPEG-1, MPEG-2, MPEG-4 and MPEG-7. Both in MPEG-1 and in MPEG-2, three different audio levels are defined. The layers are denoted by roman figures, i.e. Layer I, Layer II and Layer III. Basically, the complexity of the encoder and decoder, the encoder/decoder delay, and the coding efficiency increase when going from Layer I via Layer II to Layer III.

An MPEG audio stream consists of frames. A frame consist of a header and a data block. The data block contains the actual audio data, while the header contains information about the audio data. See figure 1.1 for an overview of the MPEG audio frame structure. The fields in the header that are relevant in this thesis are sync, layer, bit rate and frequency. The sync fields are 12 set bits in a row, and marks the beginning of a frame. The layer, bit rate and frequency fields describes which MPEG audio layer is used, the bit rate of the stream and the frequency of the audio data.

1.2.4 Java 3D

The Java 3D API[4] specification is the result of a joint collaboration between Intel Corporation, Silicon Graphics, Apple Computer and Sun Microsystems. The intended use of the API is for writing three-dimensional graphics applications and applets.

Java 3D is not an authoring environment, and does not provide built-in authoring tools. The goals of the Java 3D development has been high performance through optimizations and hardware support, rich set of 3D features, high-level, object-oriented paradigm and support for a wide variety of file formats such as VRML by run-time loaders (which are written in Java 3D).

[4] <URL:http://www.javasoft.com/products/java-media/3D/>

Up to now Java 3D have been used mostly for 3D visualization. Sun has a vision of Java 3D being used for all 3D applications, even for real time 3D games. It is not clear if the performance of Java and Java 3D is good enough for this.

Some concepts not commonly considered part of the graphics environment such as 3D spatial sound is also part of the Java 3D API. It uses the Java sound engine API in order to provide general sound and midi support.

1.3 Scalability

The definition of scalable is seemingly simple: to be able to scale. In other words: How well a solution to some problem will work when the size of the problem increases. For the kind of applications that we consider in this thesis, the problem size is the number of clients. Depending on the application, a client can either be an actual user, or just a program that is running.

There are two main aspects of scalability that has to be considered for distributed interactive applications:

1. **The scalability of the application itself.**
   The common mode of operation for interactive applications is to react on events sent from other clients. The response time for the client time should be constant with regards to the number of clients, i.e. $O(1)$. Although memory is cheap these days, the memory usage needs to be limited too. The memory requirements of the applications should also be $O(1)$, if possible.

2. **The bandwidth requirements.**
   Although basic data distribution using IP multicast scales well to large groups, there are still some issues that must be considered. One issue here is reliable data distribution. Section 2.2.5 discusses this in more detail. Another issue is the total bandwidth used, which should grow as little as possible when the number of clients is increased. Preferably the bandwidth should be constant ($O(1)$), but this is rarely possible.

It is not possible to give any firm rules when an application is scalable or not; this has to be judged from case to case. For example, an application where the bandwidth requirements are $10000 + 100 \cdot n = O(n)$, might be considered scalable although the bandwidth grows linearly with the number of clients, since the growth is small compared with total value of the bandwidth. Nevertheless, as a rule of thumb we can state that the response time, memory requirements and the bandwidth requirements should be constant, or grow with a small amount.

In this thesis, we are mainly concerned about the scalability of the applications them self. The scalability of for example the underlying network technologies such as the multicast routing protocols are only discussed when it have consequences for the scalability of the applications.

1.4 Application Level Framing

The design principle of Application Level Framing (ALF) [1] was applied during the design of these applications. The ALF principle states that an application should break the data into suitable aggregates, and the lower levels should preserve these frame

boundaries. These aggregates are called Application Data Units, or ADUs. The fundamental characteristics of the definition of the ALF design principle is that it should be possible to process each ADU as it arrives, even if the ADUs arrive out of order.

Although the original ALF paper does not mention multicast at all, Handley [6] has shown that the ALF principle is important for designing scalable applications that uses multicast. As an example, RTP was designed with multicast and ALF in mind.

In this thesis, packets and messages are used as ADUs.

1.5 The applications

The two different prototype applications might seem to be quite different, but they share several features: both uses a simple text based protocol for messages, messages are not cumulative, i.e. messages does not depend on earlier messages and a out of order messages can be processed at arrival, and the chat function for example in both application has been reused.

The main difference between the applications are that the scalability of mIR is limited by bandwidth requirements, while the scalability of m3DE is limited by the CPU usage and memory requirements of the client application. It is this difference that is the reason for developing two applications in this thesis.

Chapter 2

mIR – multicast Interactive Radio

2.1 Introduction

mIR – multicast Interactive Radio – is a suite of applications for transmitting MPEG [4] audio files using IP multicast. mIR consists of three applications: the receiver, the voting tool and the transmitter. The receiver uses an external MPEG player for actually playing out the audio. The voting tool is the application that makes mIR interactive; users can vote for songs, get more information about all the songs that are available or engage in a discussion with the other users.

All applications are written in the platform independent language Java [8] and have been tested with Sun Solaris, Linux, Windows 95 and Windows NT.

Two different "channels" are used for sending data, and each channel uses a different multicast address. On one channel (the audio channel) the actual audio data is sent, on the other channel (the information channel) information about which songs are available at the transmitter and messages to the discussion is sent. The voting tool sends information on this channel too, while the receiver only listens to the audio channel and does not send any information at all to any channel.

The focus when developing mIR has been on interactivity and scalability.

The mIR applications can be downloaded from the author's home-page[1].

2.2 The transmitter

The transmitter consists of two parts: one that receives votes from information channel, and one part that transmit the MPEG files to the audio channel. The received votes are stored in a hashtable, and these votes are used to select which song will be transmitted next. All MPEG-1 [5] audio files (layer I,II and III) can be transmitted. The audio data is multicasted using the RTP [13] protocol, using the payload type 14 for MPEG audio [12], [7]. See figure 2.1 for an overview of the functionality of the transmitter.

There is currently no graphical user interface for the transmitter; it might be available in future versions.

[1] <URL:http://www.cdt.luth.se/~rolle/mIR/>

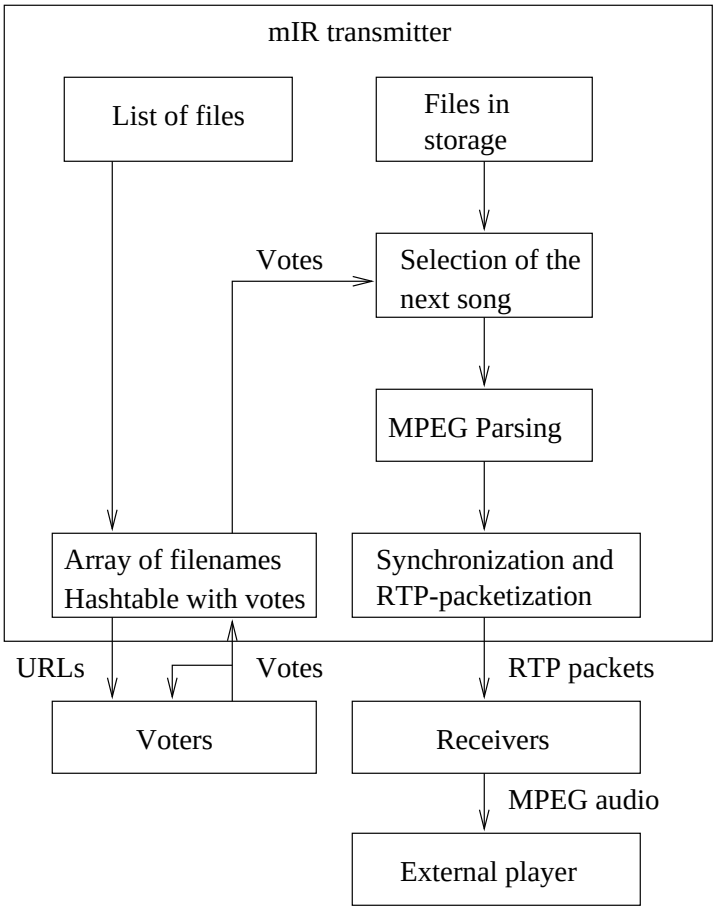

Figure 2.1: mIR transmitter

Command	Argument	Notes
“m”	The message	A message to the discussion
“c”		Clear all received votes (sent by the transmitter at start of a new song)
“v”	The song	Vote for a song
“cv”		Clear the vote from the user sending this command
“q”		Vote for “Quit transmitting this song”
“db”	An URL to the database	This URL is used in the voting tool to access the database of songs
“fu”	An URL to the filelist	The URL to the list of available songs.

Table 2.1: Available commands

2.2.1 The voting protocol

A simple text based protocol is used for voting and the discussion (see section 2.4.3). One or more commands can be sent in one RTP packet, separated by the newline character. All commands have the form “command:argument”. See table 2.1 for available commands.

2.2.2 Filenames

The URL to a file that contains a list of the songs/files that should be available for transmission are given at startup of the transmitter. The transmitter downloads the list and checks all files if they are available for reading, and removes duplicate names. A thread in the transmitter continuously sends the URL to the list of files to the information channel using the “fu” command.

2.2.3 Transmission of songs

When the transmitter has selected which song will be transmitted next, the filename is sent to a sender object that will read the file and send it to the multicast group. The marker bit in the RTP header is set on the first packet of each song. The marker bit is usually used for signaling talk-spurts when transmitting audio.

The transmission is done in a loop that consists of five steps, which will be performed until the end of the file is encountered, enough “quit” votes has been registered or an error occurs. The steps are:

1. Find next MPEG header.
   An MPEG audio file/stream consists of MPEG frames. Each frame starts with a header of four bytes, which begins with something called sync. This sync is 12 set bits in a row. The transmitter searches for the sync by reading one bit at a time from the file. If no sync is found in 1024 bits the transmission of this file is aborted.

2. Parse the MPEG header.
   When the sync is found we read the remaining bits of the header and extracts the available information (for example MPEG layer, bit rate, mono or stereo). If this was the first header of a file, information about the song is sent to the information channel using the “m” command.

3. Read the rest of the MPEG frame.
   If the header parsing was successful the rest of the MPEG frame is read.

4. Send a RTP packet to the multicast group.
   The complete MPEG frame is copied to a buffer, which is then put on the outgoing queue.

5. Synchronize.
   To ensure that the audio data is sent with the correct bit rate, the transmission is synchronized before we read the next MPEG frame. After every tenth[2] packet the transmission is synchronized against the time the transmission of this file started. The time the thread should sleep is calculated as

   $$time = n \cdot t_{frame} - (currenttime - starttime)$$

   where n is the total number of packets sent during the transmission of this file and t_{frame} is the duration of one frame. The other packets are sent with a slightly shorter interval. The reason for this is that the granularity of the system clock in Java is often as high as 10 milliseconds, and the value t_{frame} is often at around 25 milliseconds, so to avoid time drifting we need to synchronize against the start time once in a while.

After the last MPEG frame has been sent the hashtable with received votes (see section 2.2.4) is cleared, and a "c" command is sent to the information channel (the clients should now clear all data structures containing received votes). If an error have occurred during one of the steps the transmission of the file is aborted. If the transmitter have not been able to read at least 200 MPEG frames (200 frames usually represents about 5 seconds of audio) before the error occurred the file is marked as corrupt. Corrupt files will not be selected again.

2.2.4 Votes

The only commands the transmitter listen for is the "v", "cv" and "q" commands.

When a vote is received (the "v" command) an identifier for the user voting is retrieved. The RTCP [13] SDES CNAME is used for this purpose. The vote is inserted into a hashtable of received votes, using the user identifier as the key and the song as the data. This has a number of consequences:

1. Each user will only have one vote.

2. The running time of the operation is $O(1)$ (i.e. the time needed is constant with regards to the number of users and the number of available songs).

3. If two users vote for the same song, there will be two entries in the hashtable for this song.

When a "cv" command is received, the identifier for this user is retrieved and if this identifier is used as a key in the hashtable this entry is removed. Again, the running time is $O(1)$.

The "Quit transmitting this song now" (the "q" command) votes are also stored in a hashtable, using the user identifier as the key. If the size of this hashtable becomes larger than half the number of listeners, the transmitter interrupts the current

[2]The default value is every tenth packet, but this can be changed at the startup of the transmitter.

transmission and selects a new song directly. The hashtable is cleared whenever the transmission of a song starts.

The song that will be played next is selected among the songs the users have voted for by randomly selecting one entry in the hashtable. The probability for $song_i$ to be selected is thus

$$P_{song_i} = \frac{votes\ for\ song_i}{\sum_{j \in all\ songs} votes\ for\ song_j}$$

. This means that one have complete control over the selection of songs if there is only one user (the probability is then 1 for the song the user have voted for), and if there is many users and many votes there is still a chance that a song with only one vote will be selected.

After a song is selected all the votes are reset to zero. If there is no votes at all a song is selected at random.

2.2.5 Reliability of votes

To increase scalability, the received votes are the only state that the transmitter keeps. The clients are responsible for acquiring this state from the traffic on the information channel, and the transmitter does not send any information about this state to the channel.

Since neither IP multicast nor RTP provides reliable transmission, reliability must be ensured in some other way. There exist a number of solutions to this problem:

1. Accept the packet loss.
2. Transmit redundant data, for example use Forward Error Correction (FEC) or retransmit data.
3. Use Quality of Service (QoS) techniques and allocate/reserve enough bandwidth
4. Use a reliable multicast protocol.

Two of these solutions have been implemented in mIR: The use of a reliable multicast protocol and the use of redundancy.

The SRRTP solution

SRRTP (the Scalable Reliable Real-time protocol)[10] is an extension to RTP that have been developed at CDT, which implements the ideas in the SRM [3] framework. The use of SRRTP does not just solve the problem with packet loss, but also creates new ones:

1. Out of order packets.
   The number of packets that arrive out of order increases when packet loss occurs. Unless this is handled in some way, this can result in an inconsistent state at the voters and the transmitter, for example if a user changes a vote quickly and the two votes for different songs arrive out of order, the first received vote is the correct and the second should be ignored.
2. Duplicate packets.
   When a packet is lost and subsequently retransmitted, more than one copy of the packet can arrive. For votes, this would not be any problem, but for the discussion messages this would be rather annoying.

3. Late-comers.
   When voters first start the voting tool, they need to get the current state, i.e. all the votes.

4. Scalability.
   There are still questions regarding the scalability of the different reliable multicast protocols. If not even the underlying network protocols are scalable, is there any reason for the application to be scalable then?

Following the ALF design principles (See section 1.4), the receiver can handle each packet (i.e. ADU) as soon as it arrives, so the first problem was solved by simply ignoring packets that arrive to late. Removing duplicate packets provides a solution for the second problem. Since the global state is cleared after each song, the problem with late-comers can be ignored. The only consequence of this choice is that it may take longer for new clients to acquire the complete global state. Finally, since this thesis is mainly about scalability issues in applications the problem with the scalability of SRRTP has not been considered, although the protocol shows serious scalability problems (See Section 2.5.4).

The redundancy solution

The problem of packet loss can also be solved by periodic retransmission, i.e. redundancy.

In redundancy version of mIR, the votes are periodically retransmitted from the voters. The time between the retransmissions is calculated as

$$t_{sleep} = \frac{n}{k}$$

where n is the total number of members in the information channel and k is a constant (currently $k = 2$).

This means that the votes become more unreliable as the number of users increases. This might be acceptable, since these votes really are "unreliable" anyhow (a vote only guarantee that there is a chance that the song will be selected). The real consequence is that the probability that a song a user has voted for will be selected is lower than the probability calculated in Section 2.2.4, due to the fact that the probability that the vote get lost also has to be accounted for.

To avoid synchronization of votes (i.e. if two voters both vote at time t, they shouldn't both retransmit their votes at $t + t_{sleep}$ since this would create a higher load on the network and on the transmitter), a short random time is added to t_{sleep}. The discussion messages are not currently retransmitted, so the discussion must be considered as unreliable.

2.2.6 Reliability of the audio transmission

A harder problem with packet loss to solve is the problem with audio quality. MPEG-1 audio was not designed for network links with data loss, and the audio quality decreases rapidly when packets are lost. A packet loss of 1 % makes for example music encoded at 128 kbit/s almost impossible to listen to. The consequence is that a solution for the problem with packet loss is *more* important to solve for the audio transmission, since the whole system becomes impossible to use even for low ratios of packet loss.

None of these solutions described above, except the first, are trivial. Initial experiments with the use of the SRRTP protocol for the audio distribution have started.

Figure 2.2: The receiver

2.3 The receiver

The receiver can receive any MPEG audio stream that the transmitter can send, but not all RTP MPEG audio streams. The receiver fails if:

- There is more than one MPEG frame in each RTP packet.
- The MPEG frames are fragmented.

MPEG audio frames usually are smaller than 500 bytes, which means that there seldom is any need for fragmentation of frames since this is smaller than the usual MTU[3] on the internet today. Support for multiple MPEG frames in each RTP packet is planned for future versions of mIR; this would reduce the bandwidth of the audio transmission since there would be less overhead due to packet headers for each MPEG frame.

The data is sent to the player either by writing to standard input of the player process, or over an HTTP stream. Because most MPEG audio players can not handle changes in bit rate or MPEG layer in a stream, the player process is restarted when the transmission of a new song starts, i.e. when a packet where the marker bit is set is received. For users running mIR with an external player that uses a GUI, this might be an annoyance.

2.3.1 Receiving packets

When a RTP packet is received, the sequence number is checked to detect packet loss. If packet loss has occurred the count of number of packets received is adjusted so the synchronization step will carried out correctly. The MPEG frame is then retrieved from the packet, and the packet is placed in the play-out buffer. Information about the audio stream is retrieved from the MPEG frame, and the current bit rate and the MPEG layer is displayed in the user interface (see figure 2.2).

A player thread in the receiver writes the MPEG frames from the packets in the play-out buffer to an output stream. This stream is either the standard input of an external player process, or the output stream of a TCP socket. The receiver listens for HTTP connections on this socket.

After each frame has been written to the output stream the player thread is synchronized in a similar fashion as the synchronization of the transmission in the transmitter.

[3]MTU: Maximum Transmission Unit, the size of the largest packet that can be sent without fragmentation.

Figure 2.3: The voting tool

2.3.2 Performance

The receiver has been tested on a PC with a Pentium 90 MHz processor[4] with adequate results, although this highly dependent of the external player that is used. The total bandwidth required for the receiver is about 10% more than the bandwidth of the transmitted song. For MPEG 1 layer III songs encoded at 128 kbit/s the used bandwidth[5] is 142 kbit/s and for songs encoded at 112 kbit/s the used bandwidth is 125 kbit/s. A 128 kbit/s ISDN connection is thus enough for receiving and playing songs encoded at 112 kbit/s with full quality[6].

2.4 The voting tool

The voting tool is the application that makes mIR interactive. A user can see all available songs, vote for songs, communicate with other listeners and get more information about the songs. There are two different kinds of votes: votes for a specific song and "quit" votes. When the transmitter has received enough quit votes, the transmission of the currents song is interrupted and a new song is selected.

The voting tool listens for commands on the information channel and sends votes and messages from the user to the discussion to the channel (see figure 2.3).

2.4.1 Available songs

When an URL is received via the "fu" command for the first time, the filelist is downloaded, and the filenames is inserted into a hashtable (using the filename as the key). A tree structure of the filenames is then created. The user can vote for songs by selecting them in this tree structure (see figure 2.4). All votes received from other users are also available for viewing.

2.4.2 Information about songs

Information about all the available songs are kept in a SQL[9] database[7]. Currently the database contains information such as bit rate and song length. It is possible to add comments about the songs and to search the database. The function "Information about a song" in the voting tool opens a web browser with a web interface to the database, and automatically views the correct entry.

[4]Tested with both Windows 95 and Linux.
[5]Measured using IP firewall rules/accounting on a Linux workstation.
[6]This has been tested good results. All other applications that uses bandwidth must of course be closed.
[7]MySQL, <URL:http://www.mysql.com>

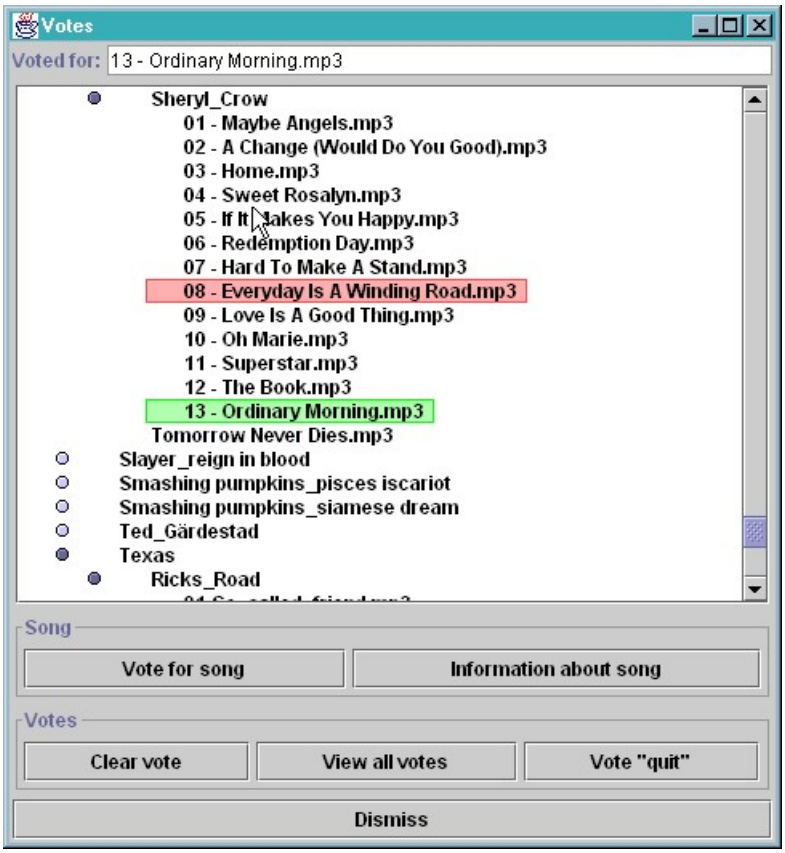

Figure 2.4: Voting

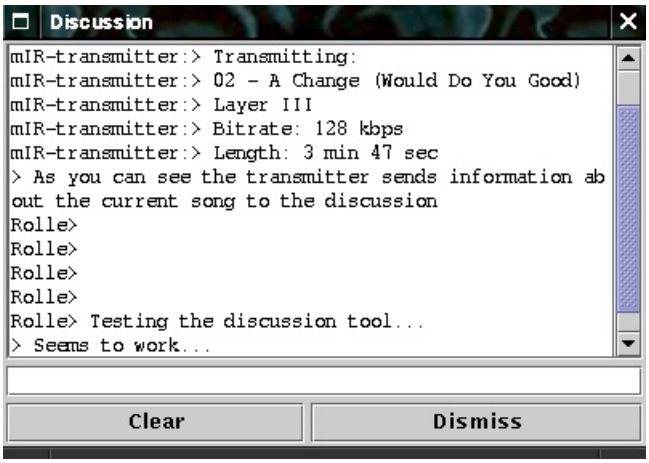

Figure 2.5: The discussion tool

2.4.3 Discussion

A simple discussion tool is included in the voting tool. Information about the current song is sent to the discussion by the transmitter. See figure 2.5 for an example view of the discussion. The messages to the discussion are not retransmitted, so some messages might not reach all other users due to packet loss.

2.5 Scalability

Both the transmitter and the voters perform operations based on commands sent to the information channel. The commands are sent by both the transmitter and the voters. These operations have to be executed quickly as incoming packets with commands may be lost otherwise. Preferably these operations should also scale well, i.e. the running time of the operations should be $O(1)$ with regards to the number of users (the time to perform an operation is constant and independent of the number of users). The memory requirements of the application should also be bounded.

2.5.1 The transmitter

As shown in section 2.2.4, the running time of all operations performed by the transmitter when commands are received from the voters is $O(1)$. The memory requirements is not constant, but grows linearly ($O(n)$) with the number of active voters (i.e. voters that actually votes). For each user voting about 100-200 additional bytes will be stored in the hashtable containing votes, so even if 10000 users are voting this amounts to 1-2 Mbytes, which is acceptable when one consider that the transmitter process usually uses 5-6 Mbytes of memory anyway. The hashtable is cleared after each song has been sent.

Although all operations can be performed in constant time, there might still be a scalability problem if all voters happen to send votes at exactly the same time. The effect to the transmitter of this is a higher rate of packet loss, but if the voters are not

completely synchronized (i.e. the next retransmission of votes do not also happen at exactly the same time) the retransmission of votes will assure that all votes are received.

2.5.2 The voting tool and the receiver

The voting tool uses the same algorithms for storing votes as the transmitter, and is therefore as scalable as the receiver is. The receiver just receives a stream of audio data with constant bit rate and sends it to an external player, so it is as scalable as RTP/RTCP.

2.5.3 Bandwidth

The SRRTP version

The total bandwidth used by mIR can be calculated as:

$$B = B_{audio} + B_{discussion} + B_{votes}$$

where B_{audio}, $B_{discussion}$ and B_{votes} is the bandwidth for the transmission of the audio data, discussion and votes respectively. The value of B_{audio} is constant for a fixed bit rate. The use of SRRTP makes $B_{discussion}$ and B_{votes} hard to calculate, but $B_{discussion}$ is usually low. B_{votes} can estimated as

$$B_{votes} = \frac{v \cdot S}{t_{song}} + B_{SRRTP}$$

where v is the total number of votes transmitted, S is the average size of one vote, t_{song} is the length of the transmitted song, and B_{SRRTP} is the overhead for SRRTP, which depends on the packet loss and the number of receivers/senders.

The redundancy version

Again, the total bandwidth used by mIR can be calculated as:

$$B = B_{audio} + B_{discussion} + B_{votes}$$

where B_{audio}, $B_{discussion}$ and $B_{votes}$is the bandwidth for the transmission of the audio data, discussion and votes respectively. The value of B_{audio} is constant for a fixed. $B_{discussion}$ is not constant, but usually quite low.

To calculate an estimate of B_{votes}, the maximum number of votes sent during the transmission of one song is calculated, and B_{votes} is then calculated as

$$B_{votes} = \frac{v \cdot S}{t_{song}}$$

where v is the number of votes sent, S is the average size of one vote and t_{song} is the length of the transmitted song. Unfortunately, v is only limited by how often users change their votes:

$$v \leq m \cdot (\frac{t_{song}}{\min(t_{sleep}, t_{song})} + C)$$

where C is the number of times one user changes the vote on average, m is the number of users voting, t_{song} is the length of the current song and t_{sleep} is as in Section 2.2.5):

$$t_{sleep} = \frac{n}{k}$$

If we only consider the case were users votes once per song ($C = 0$), then

$$v \leq m \cdot \frac{t_{song}}{\min(t_{sleep}, t_{song})}$$

If $t_{sleep} \leq t_{song}$, then

$$v \leq m \cdot \frac{t_{song}}{t_{sleep}} = m \cdot \frac{t_{song}}{(n/k)} = \frac{m}{n} \cdot k \cdot t_{song} \leq k \cdot t_{song}$$

, since $m \leq n$. The condition $t_{sleep} \leq t_{song}$ can also be written as $n \leq k \cdot t_{song}$. When these conditions hold, i.e. users don't change their votes and the total number of users is less than $k \cdot t_{song}$, then

$$B_{votes} \leq \frac{k \cdot t_{song} \cdot S}{t_{song}} = k \cdot S$$

i.e. the bandwidth is always smaller than a constant value. In the general case,

$$B_{votes} \leq m \cdot (\frac{t_{song}}{\min(t_{sleep}, t_{song})} + C) \cdot \frac{S}{t_{song}}$$

If the value of C can be considered constant, then B_{votes} grows linearly when the number of users voting increases.

2.5.4 Measurements

Instead of trying to analyze the behavior of periodic retransmission and SRRTP with different degrees of packet loss, practical experiments have been conducted. For measuring the used bandwidth a PC was configured as a firewall[8], which reported the actual value of bits/s transferred to and from a multicast address.

Table 2.2 shows the measured bandwidth. The values are the average value over 1 minute. The users were simulated with a "ghost client" application. The ghost client operate in a simple loop:

1. Send one vote and one message to the discussion.
2. Sleep for a random time. The distribution is a uniform distribution of $[0..10]$ seconds.
3. Go back to 1.

We can clearly see that the SRRTP solution is not scalable: the bandwidth increases rapidly even without packet loss when the number of users increases. The redundancy solution on the other hand behaves as predicted: it shows a small and linear growth when the number of users increases.

[8]Running the Linux operating system, <URL:http://www.Linux.org>

Users	Unreliable	SRRTP: 0%	SRRTP: 5%	SRRTP: 10%
1	1.8	1.8	1.8	1.9
10	6.1	6.2	6.5	7.2
25	13	25	32	44
50	25	105	126	136

Table 2.2: Bandwidth measurements. Values for SRRTP are packet loss in percent, for bandwidth the values are in kbit/s.

2.6 Conclusion

mIR provides a scalable system for distributing high quality audio over the internet with interactivity. Although the applications still must be considered to be prototypes, they are very robust. The transmitter for instance, has been running for over three months without any failures. Implementing the applications in Java has had a number of advantages, for example the applications are robust and platform independent and the development have been fast.

By following the ALF design principles, each incoming packet to the different application can be handled directly. Combined with efficient constant-time algorithms make the applications scalable.

Although the bandwidth is not limited, the growth is small for each additional user. The main uncertainty is the use of SRRTP, which has not yet been proven to be scalable. Another solution, the use of redundancy has though been shown to be scalable.

2.7 Future work

The main area of future work is reliability: how to provide reliability for scalable applications that uses multicast. The SRRTP protocol is not really scalable, and solutions such as redundancy are not fool-proof: we can not guarantee that all packets arrive. For applications that demand truly reliable traffic, another solution is needed.

One other interesting idea for future work is to transmit layered MPEG audio. The idea is to divide the audio data into two or more layers at the transmitter. The layers can be incrementally combined to produce higher quality audio at the receiver. Each layer would be transmitted on a different multicast group, and a receiver joins an appropriate subset of the multicast groups to receive the audio. A user could then listen to the audio at a lower quality even if it can't receive 125-145 kbit/s, which is needed now.

Most of the following ideas are planned for future versions of mIR.

- The transmitter
  - Reduce the bandwidth for audio sent by sending more than one MPEG frame in each RTP packet.
    This could reduce the bandwidth used by about 5-6 kbit/s.
  - A graphical user interface.
  - Statistics.
    The statistics could for example be inserted into the database.
  - Some sort of long term memory of votes.
    When there are no votes, a song is selected at random. Songs that have

received many votes in the past could perhaps have a higher probability of being selected again.

  - Congestion control.
    Since mIR uses quite much bandwidth, some sort of congestion control would be appropriate (in the transmitter, the receiver or in both the transmitter and receiver).

- The receiver

  - Run-time configuration of the external player.
    This would make it easier to use other players than the default ones.
  - An Applet version.

- The voting tool

  - Repeatable votes
    This could be an option such as “vote for this song until it is selected”.
  - An Applet version.

Chapter 3

The multicast 3D environment

m3DE – multicast 3D Environment – is a distributed multiuser 3D environment. Users (represented by Avatars in the virtual world) can move around in a three dimensional world. The locations of the avatars are transmitted along with other information to the other users using IP multicast. Users can also insert VRML objects into the world, and discuss with other users with the chat-boards.

The application is written in the platform independent programming language Java, and uses the Java 3D API for rendering the 3D environment. The focus has not been to create an advanced 3D environment, but to study some issues of scalability in applications for distributed 3D virtual worlds.

3.1 m3DE

The original idea was to use the new MPEG-4 standard, but due to a lack of MPEG-4 tools this idea had to be abandoned.

A first implementation of m3DE was done using a combination of Java applications and applets, a VRML viewer and the VRML External Authoring Interface[1] instead. This solution was abandoned due to performance problems. Eventually Java 3D was selected as the 3D technology instead, which provided better performance and better control.

A simple text protocol is used for sending commands among the clients. All commands are transmitted to the other clients by IP multicast. For a summary of the different commands, see table 3.1.

Multiple message boards ("chat-boards") can exist in the world. Messages to these chat-boards are sent using the "msg" command, which specifies which chat-board the message should be posted to.

The clients send their position each time the avatar moves with the "pos" command. Additional "pos" command are sent every 5:th second, to assure that new clients receive the position of the other avatars even if they do not move.

The ALF principle is used again: each packet is independent from each other and self-contained, and can be processed at arrival if packets arrive out of order. The processing of incoming packets is fast, and all algorithms are $O(1)$. In the typical case, all work that is needed when a packet arrives is to look up the user in a hashtable and change the state of the user object.

[1] <URL:http://www.vrml.org/WorkingGroups/vrml-eai/>

Command	Format	Note
“msg”	msg:chatboard:message	A message to a chat-board from a client
“pos”	pos:x:y:angle	The current position of this client
“vrml”	vrml:scale:x:y:z:ax:ay:az:URL	An URL to a VRML object and coordinates and a rotation in the world
“died”	died:	This client has closed the application

Table 3.1: Commands

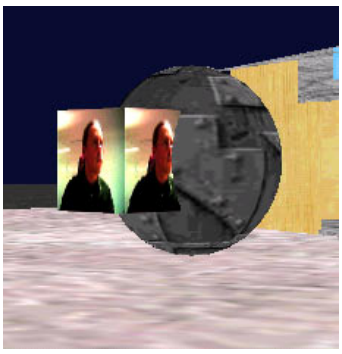

Figure 3.1: An avatar. The image is captured in real time from a camera next to the user’s computer.

The inner workings of m3DE are simple and mostly event driven. A global state exist, which contains information about all avatars such as their position and the name of the user. When an event from the GUI arrives the current state for the user is updated, and if the event was a movement the change is transmitted to the other clients. Events from the network layer is handled in an even simpler way: the state is updated with the new information. For example, when a change in position for an avatar has been received, the position of the 3D object representing the avatar is replaced with the new position. The graphics drawing is automatically handled by Java 3D.

3.1.1 The avatars

The avatars can navigate in the world in 2 dimensions and can rotate around a vertical axis going through the center of the avatar. To be able to tell who is who, the front of the avatars consists of an image. This image is downloaded using an URL to the client application, and then distributed to the other clients over the multicast channel. The URL is given at the startup of the client. This can combined with a web-cam such as WebCam32[2], and it is thus possible to incorporate a low frame-rate live video stream of each user on their avatar.

The name of user is also shown above the avatar. See figure 3.1 for an example of an avatar.

[2] <URL:http://www.kolban.com/webcam32/>

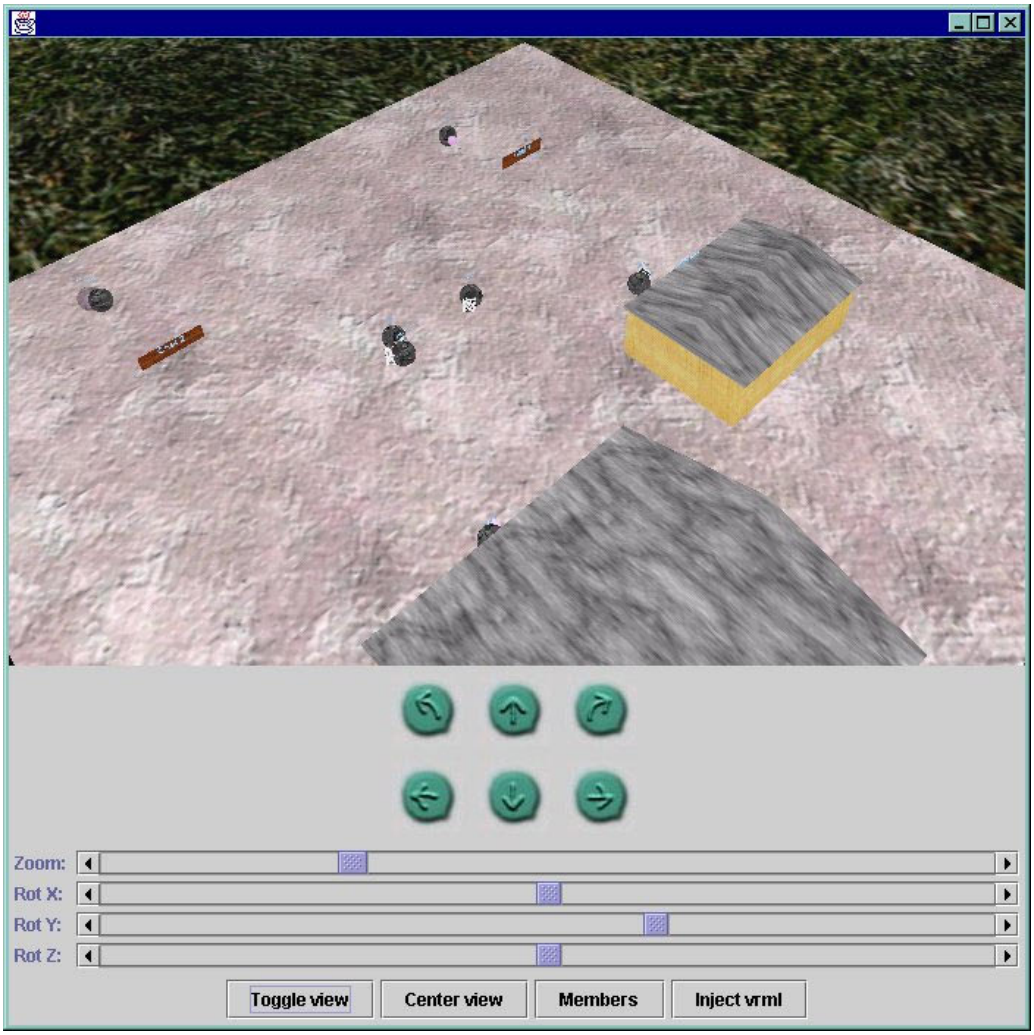

Figure 3.2: The main window. Viewed from the overview.

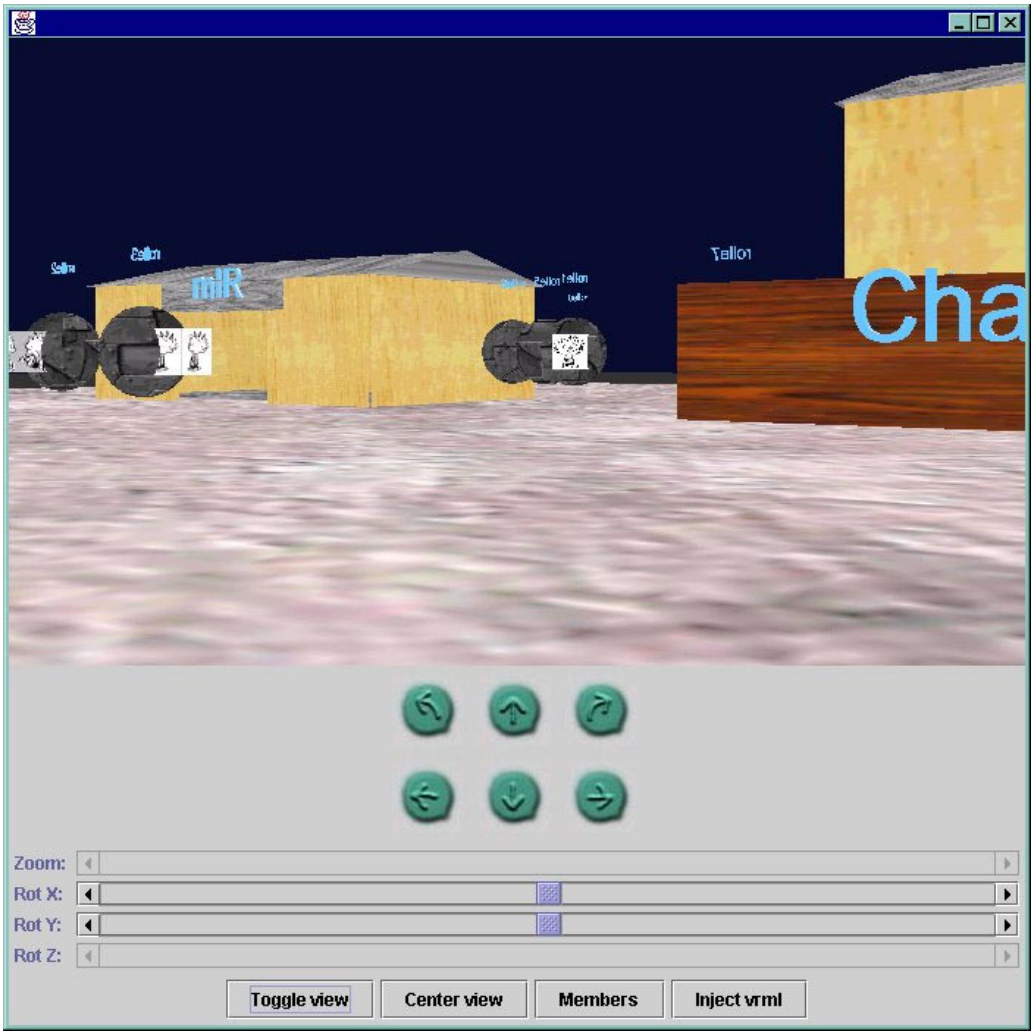

Figure 3.3: The main window. Viewed from the center of the avatar.

3.1.2 Views

A user can view the world from two different viewpoints: an overview of the complete world or from the center of the user's avatar. When viewing the world from the overview the user can rotate the world in any direction. It is also possible to change the zoom factor. From the avatar viewpoint it is possible to change the direction of the viewpoint. Examples of the two different views can bee seen in figure 3.2 and 3.3.

3.1.3 VRML objects

Users can insert VRML objects into the world. A URL to a VRML file is given, and the user then places the object in the world. The URL and the position of the object is then transmitted to the other clients using the "vrml" command. The other clients then download the VRML file and add the object to the world. Currently there is no support for telling clients that just entered the world about VRML objects that were inserted into the world earlier.

3.1.4 Actions

Certain objects have actions associated with them, such as the avatars and the chat-board. When clicking on an avatar, a dialog box with some information about that user is shown. The chat-board opens a simple chat tool. It is possible to have actions that perform arbitrary Java instructions. For instance, inside one of the houses in the current world there is an object that can start and stop the mIR application.

3.1.5 The ghost client

To allow for easy testing and debugging of the application, a ghost client were developed. This application simulates an arbitrary number of users. The "ghosts" moves randomly around in the world with one move every two seconds, and sends messages to the different chat-boards 3-4 times per minute.

3.2 Scalability

3.2.1 The application

The performance of the actual 3D display depends on the performance of the Java 3D implementation. With a modern, fast PC and a graphics card that support the OpenGL standard, the Java 3D performance is very good.

It is not the total number of objects and avatars in the world that limit the speed of the application, but the number of objects that is currently visible and moving. A complex world might actually provide better performance, since the number of visible and moving objects might be smaller (i.e. many objects are obscured by larger static objects in the world).

The memory requirements for the 3D environment currently quite high. For each avatar, 0.5-1 more Mbytes of memory is needed for m3DE. This makes the memory requirements the limiting factor of the scalability of m3DE. The speed of m3DE on a fast PC is still acceptable when the memory usage become intolerable[3].

[3]This level is reached for about 30-35 users.

3.2.2 Bandwidth

Each client sends the position of the avatar each time the avatar moves, plus once every 5:th second to make sure late-comers receives the other avatars' coordinates sometime even if they do not move. If an avatar moves in average once per second, then the bandwidth used for position reporting is about 1 kbit/s per avatar.

The continuously updated images use most of the bandwidth requirements of m3DE. The actual bandwidth depends on the image size and the update frequency. It is advisable to reduce the update frequency when the number of users increases.

If the application can be made scalable enough to make the bandwidth the limiting factor, there are some possible optimizations. One approach would be to some sort of "layered encoding": the world is divided into areas, and position reports for each part are sent on a different multicast channel. A client will then listen to only the appropriate channels: one for global announcements and the channel for the area where the clients is currently located. Since the images are currently transmitted uncompressed, much could be gained by some compression methods.

Currently the VRML files used when inserting VRML objects in the world are downloaded from a HTTP server, but for the HTTP servers in use today this is not a problem for the number of users the application itself can handle. A completely multicast based solution such as the solution for the images would be preferable, and is planned for future versions.

3.3 Conclusions

Again, the use of the ALF design principle gives a good framework for a scalable application. Unfortunately, m3DE cannot be called scalable; the memory requirements limit the scalability too much. Still, the application is usable for up to 30-35 users, which is rather good for a "virtual world" type of application.

A not so surprising conclusion is that the scalability of m3DE, in contrast to mIR, is limited by the application itself and not the bandwidth requirements.

3.4 Future work

The are two areas of future work:

1. More work regarding the study of scalability issues with these kinds of applications, and
2. Future work with the prototype application.

3.4.1 Scalability

The main area for future work is of course to try to limit the memory usage. It is not clear what actually can be done, since displaying 3D graphics is processing and memory intensive. A solution might be to try to limit the number of avatars in any given view of the world, without limiting the total number of avatars in the world.

3.4.2 The application

For the application itself, the amount of work that could be done is practically unlimited. These are some of the more interesting ideas:

- Make the application completely distributed, i.e. remove the use of HTTP servers for distributing VRML objects.
- Persistence: Changes made to the world should not disappear if all users quit the client applications.
- Implement collision detection so the avatars can't move through objects and walls.
- More advanced avatars: arbitrary VRML objects instead of the predefined model, emotions such as happy, sad, waving etc.
- Movement in three dimensions.
- 3D Audio.

Chapter 4

Conclusions

There are clear differences in the scalability of mIR and m3DE. For mIR, the main concern for scalability is the bandwidth requirements, which can be solved by not using reliable multicast and providing reliability with redundancy instead. For m3DE it is the memory usage and the performance of the 3D display that are limiting the scalability.

The ALF principle has shown to be an important help when designing interactive applications that uses multicast (or any network protocol for that matter). ALF is not enough, the algorithms for processing incoming packets has to be efficient and scalable too.

Further work is necessary in the area of reliability for applications that uses multicast and need to be scalable. Much research is currently being conducted in this area. For the applications, much future work can be done, especially for m3DE. One area that has been avoided in this thesis is the question of usability of 3D applications.

The goals have been fulfilled: some problems of scalability for interactive applications have been identified, and some of them have been solved. Most of the original ideas for mIR have been implemented, and a basic 3D interactive environment have also been implemented.

Appendix A

Glossary

ADU Application Data Unit

ALF Application Layer Framing

API Application Programming Interface

HTTP Hypertext Transfer Protocol

IP Internet Protocol

mIR multicast Interactive Radio

m3DE multicast 3D Environment

MPEG Motion Picture Expert Group

RTCP RTP Control Protocol

RTP Real Time Protocol

SSRC Synchronisation Source

TCP Transmission Control Protocol

UDP User Datagram Protocol

URL Uniform Resource Locator

VR Virtual Reality

VRML Virtual Reality Markup Language

Bibliography

[1] D. Clark and D. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of SigComm*, pages 201–208, 1990.

[2] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[3] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application framing. In *ACM SIGCOMM*, 1995.

[4] MPEG Group. <URL:http://cselt.stet.it/mpeg/>.

[5] MPEG Group. ISO/IEC International Standard 11172; coding of moving pictures and associated audio for digital storage media up to about 1,5 mbit/s, 1993. <URL:http://cselt.stet.it/mpeg/>.

[6] M. Handley. On Scalable Internet Multimedia Conferencing Systems, 1997.

[7] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar. RTP payload format for MPEG1/MPEG2 video, 1998. IETF RFC2250.

[8] JavaSoft Inc. The Java Language. <URL:http://www.javasoft.com/>.

[9] ISO/IEC 9075:1992. Information technology – Database languages – SQL, 1992.

[10] P. Parnes. Scalable Reliable Real-time Transport Protocol - SRRTP. Work in progress[1], 1996.

[11] J. Postel. User Datagram Protocol, 1980. IETF RFC768.

[12] H. Schulzrinne. RTP profile for audio and video conferences with minimal control, 1996. IETF RFC1890.

[13] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, 1996. IETF RFC1889.

[1] <URL:http://www.cdt.luth.se/~peppar/docs/rtp_srm/>