multicast Interactive Radio

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

Phone: +46-920-72887
E-mail: rolle@cdt.luth.se
Home page: http://www.cdt.luth.se/˜rolle/

March, 1999

Abstract

This document describes multicast Interactive Radio, a system for distributing high quality audio over the Internet with interactivity. The system is implemented in the platform independent language Java, and uses IP multicast for distributing the audio.

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 are 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].

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

1.1 Technical background

1.1.1 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 hosts identified by a single IP destination address. The datagrams are 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.

1.1.2 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.

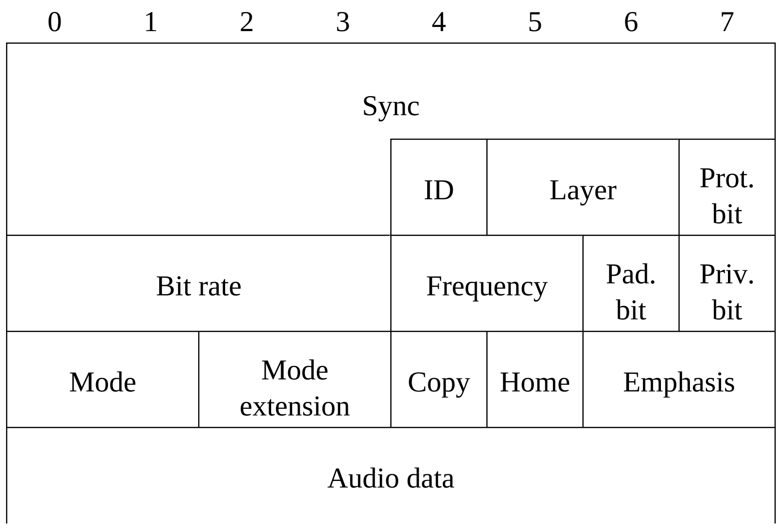


Figure 1: The MPEG Audio header

1.1.3 MPEG-1 Audio

MPEG is a working group in a subcommittee of ISO/IEC 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 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 mark the beginning of a frame. The layer, bit rate and frequency fields describe which MPEG audio layer is used, the bit rate of the stream and the frequency of the audio data.

1.2 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 report, 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 have 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 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 2.1.5 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.

1.3 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 report, packets and messages are used as ADUs.

2 mIR – multicast Interactive Radio

2.1 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 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.

2.1.1 The voting protocol

A simple text based protocol is used for voting and the discussion (see section 2.3.3). 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 1 for available commands.

2.1.2 Filenames

The URL to a file that contains a list of the songs/files that should be available for transmission is given at the startup of the transmitter. The transmitter downloads the list and check all files if they are available for reading, and removes duplicate names. A thread

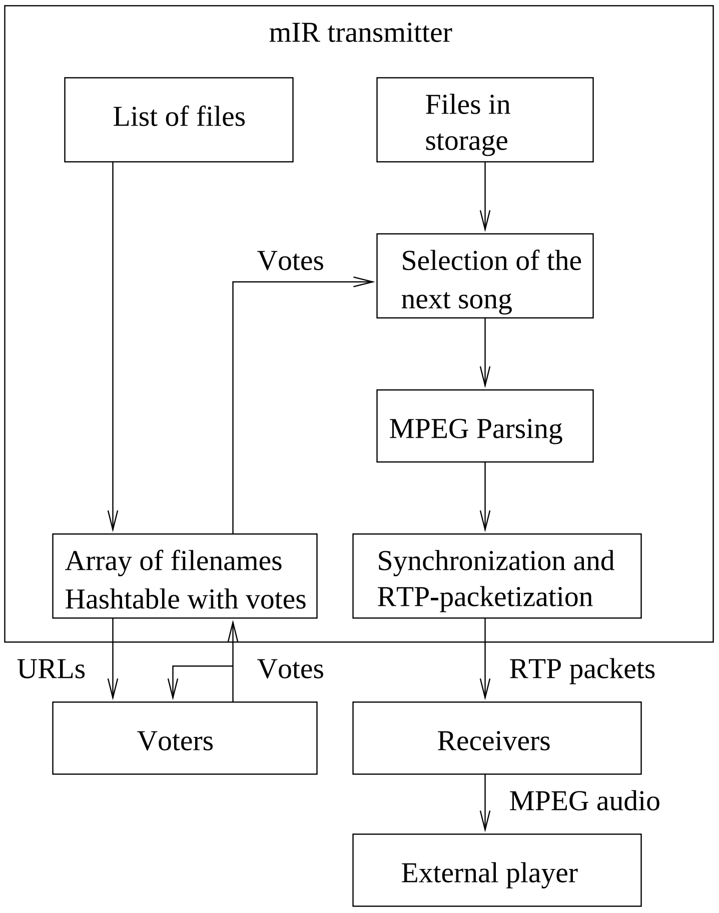


Figure 2: mIR transmitter

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 1: Available commands

in the transmitter continuously sends the URL to the list of files to the information channel using the “fu” command.

2.1.3 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 have 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

   $$time = n \cdot t_{frame} - (current\,time - start\,time)$$

   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 of received votes (see section 2.1.4) 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 has occurred

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

during one of the steps the transmission of the file is aborted. If the transmitter has 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.

2.1.4 Votes

The only commands the transmitter listens for are 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 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 has complete control over the selection of songs if there is only one user (the probability is then 1 for the song the user has voted for), and if there are 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 are no votes at all a song is selected at random.

2.1.5 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 has been developed at CDT, which implements the ideas in the SRM [3] framework. The use of SRRTP does not just solve the problem of 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 one 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?

Following the ALF design principles (See section 1.3), 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.

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

In the 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 guarantees 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 2.1.4, due to the fact that the probability that the vote gets 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.

2.1.6 Reliability of the audio transmission

A harder problem to solve with packet loss is the problem of 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, is trivial. Initial experiments with the use of the SRRTP protocol for the audio distribution have started.

2.2 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.

2.2.1 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 the number of packets received is adjusted so the synchronization step will be carried out correctly. The MPEG frame is then

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

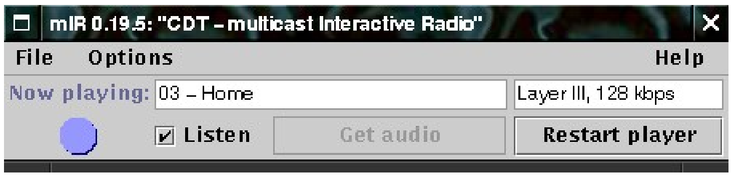


Figure 3: The receiver

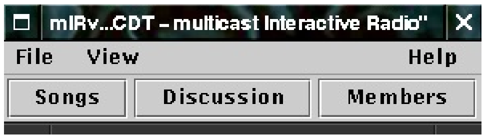


Figure 4: The voting tool

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 3).

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.

2.2.2 Performance

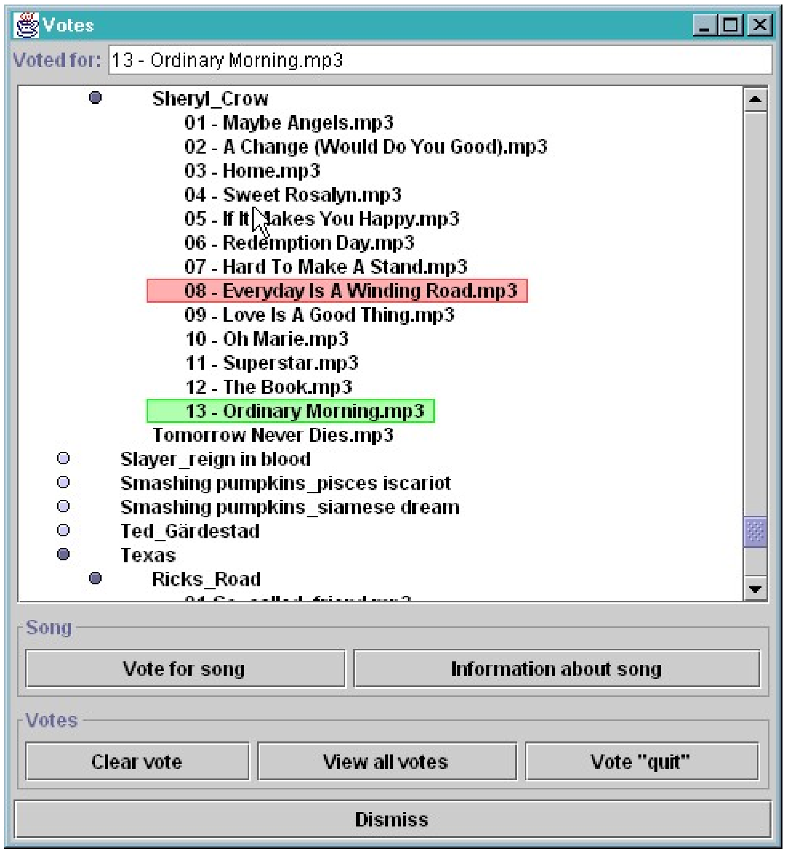
The receiver has been tested on a PC with a Pentium 90 MHz processor[4] with adequate results, although this is highly dependent on the external player that is used. The total bandwidth required for the receiver is about 10% more than the bandwidth of the transmitted song. For MPEG 1 layer III songs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

2.3 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 4).

[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.



Voted for: 13 - Ordinary Morning.mp3

```
    ● Sheryl_Crow
        01 - Maybe Angels.mp3
        02 - A Change (Would Do You Good).mp3
        03 - Home.mp3
        04 - Sweet Rosalyn.mp3
        05 - If It Makes You Happy.mp3
        06 - Redemption Day.mp3
        07 - Hard To Make A Stand.mp3
        08 - Everyday Is A Winding Road.mp3
        09 - Love Is A Good Thing.mp3
        10 - Oh Marie.mp3
        11 - Superstar.mp3
        12 - The Book.mp3
        13 - Ordinary Morning.mp3
      Tomorrow Never Dies.mp3
  ○ Slayer_reign in blood
  ○ Smashing pumpkins_pisces iscariot
  ○ Smashing pumpkins_siamese dream
  ○ Ted_Gärdestad
  ● Texas
    ● Ricks_Road
```

Figure 5: Voting

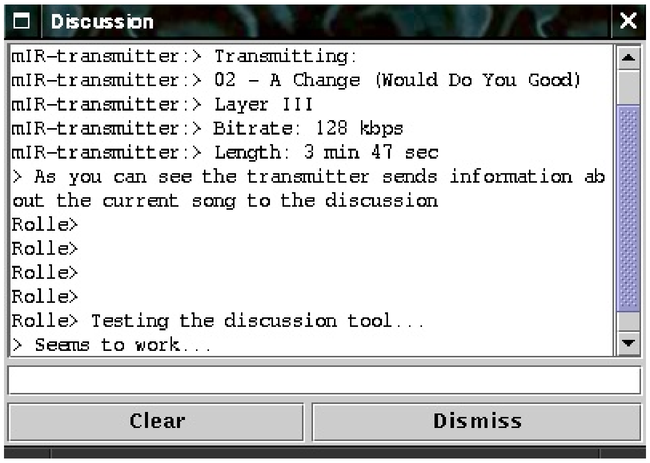
2.3.1 Available songs

When an URL is received via the ”fu” command for the first time, the filelist is downloaded, and the filenames are 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 5). All votes received from other users are also available for viewing.

2.3.2 Information about songs

Information about all the available songs is 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.

[7]MySQL, <URL:http://www.mysql.com>



```
mIR-transmitter:> Transmitting:
mIR-transmitter:> 02 - A Change (Would Do You Good)
mIR-transmitter:> Layer III
mIR-transmitter:> Bitrate: 128 kbps
mIR-transmitter:> Length: 3 min 47 sec
> As you can see the transmitter sends information ab
out the current song to the discussion
Rolle>
Rolle>
Rolle>
Rolle>
Rolle> Testing the discussion tool...
> Seems to work...
```

Figure 6: The discussion tool

2.3.3 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 6 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.

2.4 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.

2.4.1 The transmitter

As shown above (see section 2.1.4), the running time of all operations performed by the transmitter when commands are received from the voters is $O(1)$. The memory requirement is not constant, but grows linearly ($O(n)$) with the number of active voters (i.e. voters that actually vote). For each user voting about 100-200 additional bytes will to 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 considers 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 at 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 does not also happen at exactly the same time) the retransmission of votes will assure that all votes are received.

2.4.2 The voting tool and the receiver

The transmitter 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.

2.4.3 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 be 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, 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:

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

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 above (see Section 2.1.5).

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

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

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

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,

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

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

2.4.4 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 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>

3 Related work

There exist a few other programs for distributing high quality audio over the Internet. A regular WWW server can also be used for this, both for "asynchronous" listening (first the song has to be downloaded, then it the song is played) and for streaming. No publically available systems that incorporates interactive parts are known by the author.

3.1 Real

The Real Server and the Real Player[9] are often used for transmission of audio, but the quality is far from the quality one can get from MPEG audio. Both unicast and multicast can be used.

3.2 Shoutcast and Icecast

Although unicast solutions have existed for a long time, it is the recently released Shoutcast[10] that is most well known.

Shoutcast encodes the audio currently being played through the sound card of the computer as MPEG audio and transmit this data to a server program that handles the distribution of the music. The server program can be running on the same workstation or on a separate server, and works much like a simple WWW server.

Since Shoutcast uses unicast it is not very scalable. The bandwidth usage can be calculated as

$$B \geq n \cdot bitrate$$

where n is the number of listeners and $bitrate$ is the bit rate of the song currently being transmitted.

A similar system is Icecast[11], which was created as an free GPL alternative to Shoutcast that could run on Linux and other Unix platforms.

3.3 liveCaster

Ross Finlayson's liveCaster[12] program is similar to mIR, but does not contain any interactive parts. Since liveCaster transmits MPEG Audio files from storage using RTP over IP Multicast, mIR and liveCaster is compatible. liveCaster can also do transcoding of MPEG streams to lower bit rates.

4 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 has been fast.

[9] <URL:http://www.real.com/>
[10] Released in December 1998: <URL:http://www.shoutcast.com/>
[11] <URL:http://icecast.linuxpower.org/>
[12] <URL:http://www.live.com/liveCaster/>

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.

5 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 a considerable amount of 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.

References

[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[13], 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.

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