On large scale real time music distribution: security, reliability and heterogeneity

Roland Parviainen

Division of Software Engineering
Department of Computer Science and Electrical Engineering
Luleå University of Technology
S-971 87 Luleå
Sweden

October 2001

Supervisor

PhD Peter Parnes, Luleå University of Technology

Abstract

This thesis presents a system for real-time large scale distributed music distribution, such as radio on the Internet. The existing technologies do not enable the full potential that broadcast media on the Internet can give, such as a very large number of interactive receivers. A technology that can help achieve potential is IP-multicast, which enables broadcast applications to scale to millions of receivers. However, IP-multicast creates new problems that have to be solved for it to be used on the Internet; some of these problems are addressed in this thesis.

The system presented in this thesis, mIR - multicast Interactive Radio, has been used as a prototype to implement and examine different solutions to some of the issues of IP-multicast and real time music distribution. These are problems such as heterogeneity, reliability and security.

Heterogeneity and reliability are different aspects of the same problem: that IP-multicast is an unreliable Internet transport mechanism. In this thesis different methods for avoiding these problems are described and evaluated.

Security in the context of IP-multicast has many aspects. While the problems of confidentiality and authentication have been extensively examined, there are still a few unresolved problems. One of these problems is traitor tracing: how to know the origin of illegal copies of a media object such as a radio transmission. An application of digital watermarking, fingerprinting, can be used to distinguish between different copies of the same media object, but requires that each copy is individually marked. This contradicts the basic property of multicast that ensures its scalability: that everyone receives exactly the same data. We show how we can combine the different goals of fingerprinting and IP-multicast while still maintaining the scalability features of multicast.

Contents

Publications

This thesis consists of an introduction and three papers, of which two have been published and one has been submitted for review. I am the main author of all papers and have also done all work described in the thesis. The papers are:

Paper 1 Roland Parviainen. Multicast Interactive Radio. In *Proceedings of The First International Conference and Exhibition on The Practical Application of Java*, London, UK, April 1999, pp. 137-153.

Paper 2 Roland Parviainen, Peter Parnes. Error resilience methods for multicast MPEG-1 Audio transmission. Submitted to the *Networking 2002, the second IFIP TC6 Networking conference*.

Paper 3 Roland Parviainen, Peter Parnes. Large scale distributed watermarking of multicast media through encryption. In *Proceedings of the International Federation for Information Processing, Communications and Multimedia Security Joint working conference IFIP TC6 and TC11*, Darmstadt, Germany, May, 2001, pp. 149-158.

Acknowledgments

First, I would like to thank my supervisor Peter Parnes. It was Peter who wrote the first basic parts of mIR and who suggested that I should continue working on mIR.

I would also like to thank my collegues at CDT and PVT, especially those who read and had comments on earlier versions of this thesis and the papers in it.

Finally, I would like to thank my family and friends who made sure that not all my time have been spent i front of computers.

Luleå, October 2001
Roland Parviainen

Thesis Introduction

Introduction

The growth of real time music distribution on the Internet has increased rapidly in the last year; for example the Live365[1] site have more than 30,000 different radio stations on-line that users can listen to for free. The potential audience for a radio station be from a few thousand (a small local station) up to several million people by broadcasting on the Internet. This potential is never reached, main due to a scalability problem inherent in current technology used by almost all media broadcasts today: the use of a point to point technology. The resources needed by the broadcaster (the server, sender etc.) increases linearly with the number of receivers (clients, users etc.), which creates large demands on network infrastructure, computer hardware, and so on. With a broadband connection of 10 Mbit/s, it is only possible to support from 50 to 150 listeners with near CD-quality audio.

One alternative approach is IP multicast, a technology were the resources needed by the broadcaster is similar to the resources needed by a receiver. That is, if we have resources to receive a stream we can also broadcast a stream to a large audience. This technology introduces several new problems that have to be solved if it ever will be used in any larger scale. In this thesis some of these problems are considered.

Background

The work described in this thesis started as a small project; to finish an existing application called mIR - multicast Interactive Radio. Contrary to the name, mIR was not interactive at all, it was just a "radio" application using the IP multicast protocol. It also had some other limitations; for example it could only use a certain kind of files, used a non-standard protocol, etc. The work on mIR, adding interactivity, standard compliance, etc. led to the first paper, "multicast Interactive Radio". Interactivity in this thesis relates to interactivity between different users, not between a user and computer programs.

In that paper several problems were identified. For example, the system suffered from bad error resilience: if some data was lost in the network the audio quality at the receiver

[1] <URL:http://www.live365.com/>

degraded quickly. The chat messages where unreliable, there was no congestion control, etc. It was decided that work should continue on these and other related problems.

The goal was to create a system that was scalable, interactive, reliable, secure and capable of handling users with different bandwidths and packet loss ratios.

Real time media

The focus in this thesis is on real-time streaming media, not media-on-demand and similar systems. It is applications such as radio and TV broadcasts on the Internet and video conferencing, not systems such as media-on-demand that is studied.

These applications have certain common properties:

- Real-time
  - Need sufficient bandwidth, can not wait for a download to finish
  - Retransmissions are often not feasible
- Large scale
  - Radio is a typical situation when we want to support a large number of listeners
- Every one receives to the same media
  - The same data should be sent to every receiver

Most current systems use a technology called unicast, where the source (or server, sender) sends a new copy of the same data to everyone. If a thousand people listen to the same broadcast, the source has to send a thousand copies. This does not scale, that is, when the number of listeners increases the source and the network quickly becomes overloaded. To be able to broadcast to many listeners expensive Internet connections and computers are necessary.

The technology behind mIR is multicast, which fits naturally to the radio paradigm and the properties described above: the source just sends one copy of the data to the "ether", in this case the network. The listener tune in to the correct frequency, or the correct multicast group. Contrary to the traditional radio no license to transmit data is necessary, anyone can have their own radio station as long as they have rights to the media being broadcasted. Combined with the property that the source have to send only one copy of the media stream it becomes feasible for even home users to transmit media to a large audience.

The multicast technology is not for free though; there are several problems and limitations that have to be considered. Some of them are listed below:

Reliability There is no reliability provided by the network layer. If some data are lost in the network it will not be automatically retransmitted. The applications must take this into consideration and adapt accordingly.

Security Security requirements depend on the intended application. For example, a commercial transmission might have requirements such as strong authentication and encryption while other applications might require source authentication and data integrity protection. Often so called "traitor tracing" is wanted; if a stream is used in illegal ways we should be able to trace who the responsible user is.

Heterogeneity A basic property of the Internet is heterogeneity; the resources available to different users such as bandwidth vary widely. Preferably we would like to support both users with high bandwidth links and users with low bandwidth and/or high packet loss ratios. Heterogeneity is related to reliability, in the sense that if a reliable system can handle users with different amounts of packet loss.

The first two problems are the focus of the remaining papers in this thesis. All results presented have been implemented in mIR, which by now have transformed from a relatively simple application to a quite advanced research prototype. From a user's perspective not much has changed, since most new and more advanced features are implemented in the layers below the user interface.

Distributed watermarking

Purely software based copy protection and prevention schemes have never worked. Every proposed solution have been cracked and most security experts believe it is impossible to create a system that can not be attacked. It does not matter that only an expert would be able to crack a system, since it is always possible to create a simple "point and click" application that even novice computer users can use and spread over the Internet.

Instead we could try to discourage copying, by being able to find the user who made and spread an illegal copy. Fingerprinting is one way of achieving this. Fingerprinting is an application of digital watermarking, a technology that tries to embed information in a media object that cannot be removed without destroying the media object itself. In fingerprinting each copy of a media object has a unique watermark, linked to for example the identity of the license owner of that particular object. If an illegal copy is found, the fingerprint is extracted and we get an indication of who made the copy.

The goal of fingerprinting is contrary to the goal of multicast: in fingerprinting every user should receive a different, unique copy of for example a radio broadcast while in multicast we send the exact same data to each listener. In the paper "Large Scale Distributed Watermarking of Multicast Media through Encryption" (See page 39) a way of combining these two goals is presented. By using encryption we can, for a cost of doubling the used bandwidth, ensure that each user receives a unique fingerprinted stream.

These solutions do not discourage or prevent larger groups of pirates or professional pirate copy facilities, but that is not the goal either.

The issues of encryption and authentication, which also are very important security requirements are not discussed in this thesis. A lot other work in this area has been done; see for example [2][5][6][30][29].

Reliability

In real-time streaming applications there is often no time to retransmit a lost packet; when the retransmitted packet is finally received it might already be too late to be of any use. Instead we must make sure that the application can handle a certain amount of loss. The audio codec used in mIR is MPEG-1 Layer III, known as "mp3" [14], which is probably the most common audio codec used on the Internet. It can support near CD-quality audio at a compression rate of 11:1. The current standard for transmitting mp3 audio over RTP/IP multicast does not handle lost packets very well. Streams with a packet loss ratio of about 2% are rendered with very bad audio quality.

The problem of handling packet loss has been extensively studied when it comes to conferencing audio and IP telephony, but this work has always focused on speech, which differs from high fidelity music.

In the second paper "Error resilience methods for multicast MPEG-1 Audio transmission" (see page 29), we describe and compare several ways to increase the error resilience of mIR. Forward Error Correction, FEC, is a way of adding redundancy to a stream in a way so that we use the redundant information to recover the lost packets. Two different FEC methods were implemented and compared.

It is also possible to rearrange the data in the stream to increase the error resilience, for example by interleaving packets or changing the packetization format. The format for packetization of MPEG-1 audio described in RFC2250 [18] is sub-optimal for mp3, due to the way data is stored in the MPEG stream.

A mp3 audio stream consist of a stream of frames after each other. Each frame has two parts, the header and the data. In MPEG-1 Layer I and II all data needed to decode a frame is stored in the data portion of that frame, but in Layer III some data can be stored in the previous frames as well. If one frame is lost, me might loose data from several other frames which thus are rendered useless. We show that by rearranging the stream according to RFC3119 [11] so that all data for a frame are sent together as one packet the audio quality when packet loss occurs improves substantially.

Current status of mIR

The mIR prototype is the basis for all the work in this thesis: all described algorithms and protocols are implemented in mIR which is downloadable from the authors home page[2].

Several parts of mIR are based on the Marratech Pro application, a commercial product from Marratech AB that is the result of earlier research at the Software Engineering division. These parts are the RTP network stack, the SAP[17] and SDP[16] protocol implementations and the chat component.

The RTP stack has been heavily modified, where support for forward error correction is the largest modification. The support for forward error correction is based on "An RTP Payload Format for Generic Forward Error Correction" [28], but an additional similar protocol

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

was created and implemented that uses Reed-solomon codes instead of parity as the error correction code. To test the error resilience the network layer can either use a simulated packet loss (either uniformly distributed or generated by a 2-stage Markov chain) or use a trace of earlier measurements of real packet loss.

Encryption and the watermarking algorithm presented in the third paper are implemented but is not enabled in the publically released version due to the limitations discussed in the paper.

Interactivity is still a bit limited. The chat functionality is now provided by a module from the Marratech Pro product and uses a reliable multicast protocol, SRRTP [23] that is much more advanced than the old chat tool. This is one of the more directly noticeable changes in mIR compared to what is described in the first paper. The voting mechanisms described in the first paper have not noticeably changed.

Most of the initial goals have been met; in the first paper in the thesis we show that it is scalable and interactive, in the second paper we show different ways of achieving reliability and finally in the last paper we discuss the security aspects. One goal remains: to be able to handle users with different amounts of available bandwidth in the same session.

Future work

Although mp3 is very popular, it has a few drawbacks: it only handles one or two channel sound, there are high licensing fees, newer codecs have a higher compression efficiency, etc. One codec that will be supported in mIR in the future is the new Ogg Vorbis[3] codec. It does not have these limitations and will also enable new advanced features such as layered multicast. Layered multicast will enable us to achieve the final goal, to be able to handle users with different amounts of available bandwidth. Initial prototyping that show that this is possible have been done.

There is much room for improvement of the interactive parts of mIR, such as more advanced voting features. Adding support for video, both as a broadcast media and as a interaction media (together with chat and audio for videoconferencing) is also a possible future development of mIR.

[3] <URL:http://www.xiph.org/ogg/vorbis/>

Part 1

Multicast Interactive Radio

Multicast Interactive Radio

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

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.

1.1 Introduction

mIR – multicast Interactive Radio – is a suite of applications for transmitting MPEG [13] 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 [19] 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/>

1.1.1 Technical background

IP multicast

IP multicast [9] 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.

Real-time Transport Protocol

The Real-time Transport Protocol, RTP [27], 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 [25]. 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.

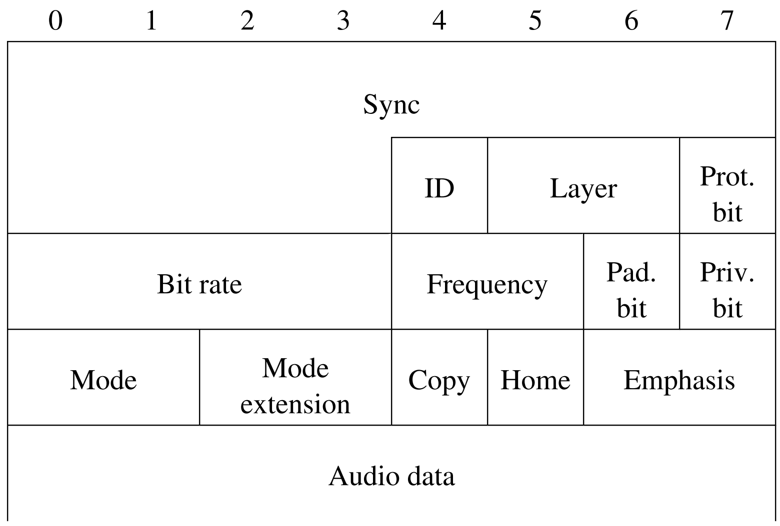

Figure 1.1: The MPEG Audio header

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 1.1 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.

1.1.2 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 1.2.1 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.

1.1.3 Application Level Framing

The design principle of Application Level Framing (ALF) [8] 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 [15] 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.

1.2 mIR – multicast Interactive Radio

1.2.1 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 [14] audio files (layer I,II and III) can be transmitted. The audio data is multicasted using the RTP [27] protocol, using the payload type 14 for MPEG audio [26], [18]. See figure 1.2 for an overview of the functionality of the transmitter.

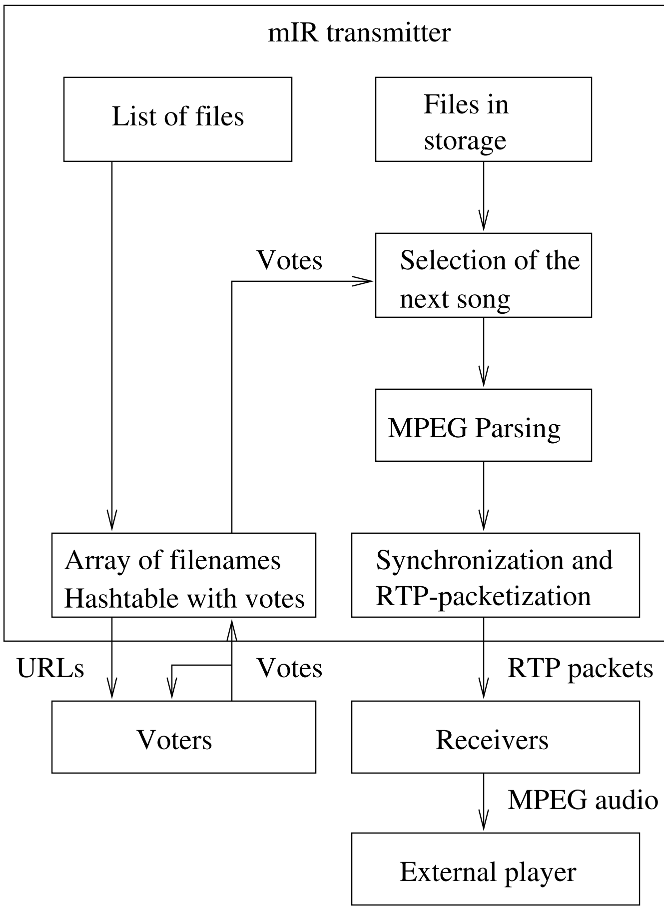

Figure 1.2: mIR transmitter

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 1.1: Available commands

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

The voting protocol

A simple text based protocol is used for voting and the discussion (see section 1.2.3). 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 1.1 for available commands.

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

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 1.2.1) 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 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.

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 [27] 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:

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

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.

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)[23] is an extension to RTP that has been developed at CDT, which implements the ideas in the SRM [12] 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 1.1.3), 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 1.2.1, 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.

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.

1.2.2 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.

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 syn-

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

Figure 1.3: The receiver

chronization step will be 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 1.3).

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.

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].

1.2.3 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 1.4).

[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.

Figure 1.4: The voting tool

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 1.5). All votes received from other users are also available for viewing.

Information about songs

Information about all the available songs is kept in a SQL[21] 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.

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 1.6 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.

1.2.4 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.

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

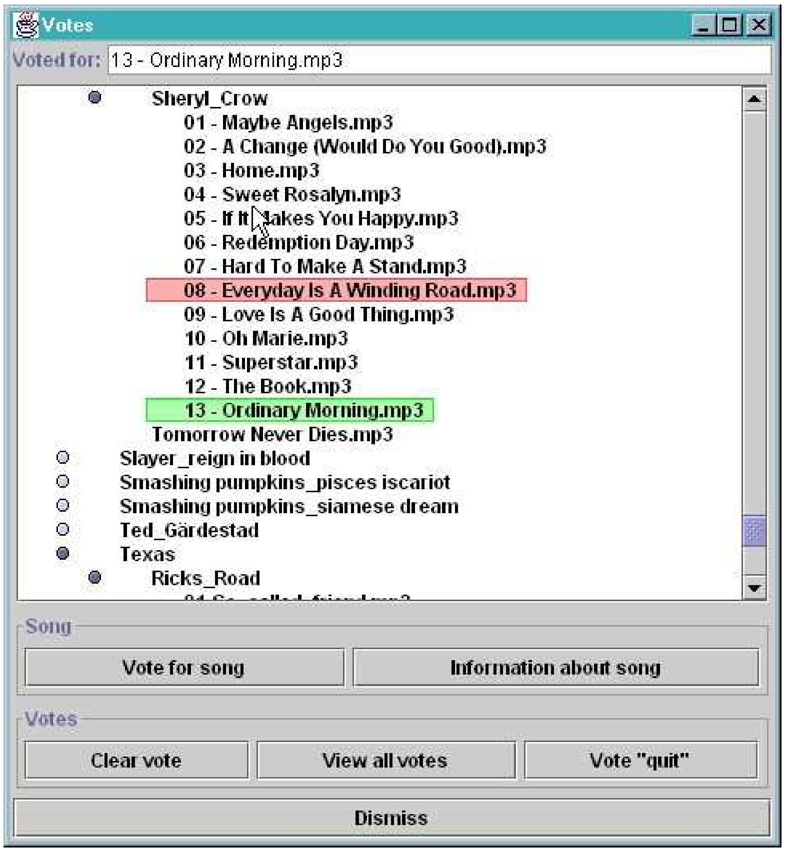

Figure 1.5: Voting

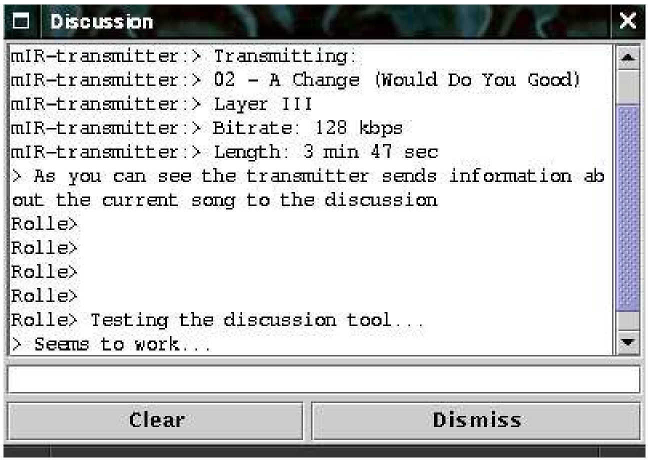

Figure 1.6: The discussion tool

The transmitter

As shown above (see section 1.2.1), 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.

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.

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 1.2.1).

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

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

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

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 1.2 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.

1.3 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.

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

1.3.1 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.

1.3.2 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.

1.3.3 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.

1.4 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.

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.

[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/>

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.

1.5 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.

Part 2

Error resilience methods for multicast MPEG-1 Audio transmission

Error resilience methods for multicast MPEG-1 Audio transmission

Roland Parviainen, Peter Parnes
Department of Computer Science/Centre for Distance-spanning Technology
Luleå University of Technology, 971 87 Luleå, Sweden
Roland.Parviainen@cdt.luth.se, Peter.Parnes@cdt.luth.se

Abstract

An increasingly popular method of broadcasting high quality audio such as music on the Internet is IP multicast and MPEG-1 Layer III audio, well known as "mp3" audio. The main problem with this method is packet loss, which MPEG Layer III was not designed to cope with. This paper presents an evaluation of a number of different methods to alleviate this problem. All these solutions have been implemented in the application mIR and the mStar framework.

We show that a new packetization format for MPEG Layer III provides better error resilience than the standard format, and that the use of forward error correction can be used to further increase the error resilience by reducing the packet loss at the receiver.

2.1 Introduction

Using IP multicast to broadcast music to a large audience is a promising application due to the scalability of multicast. The MPEG-1 Layer III audio compression format is by far the most common format for music on the Internet and more and more popular for real-time audio streaming. Several tools for MPEG-1 audio transmission over IP multicast exist such as mIR[1] and liveCaster[2] All these applications perform badly when packet loss occur due to MPEG-1 Layer III's bad error resilience, although the experimental new payload format in liveCaster is an improvement. In this paper we evaluate several methods implemented in mIR to alleviate this problem.

2.1.1 IP multicast and RTP

IP multicast [9] 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.

[1] <URL:http://www.cdt.luth.se/ rolle/mIR/>
[2] <URL:http://www.live.com/liveCaster/>

The Real-time Transport Protocol, RTP [27], 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 [25]. RTP provides no additional reliability and assumes that low levels of loss are acceptable for audio and video applications.

2.1.2 MPEG-1 Audio

An MPEG-1 [14] audio stream consists of frames with a header and a data block. The data block contains the actual audio data, while the header contains information about the audio data.

2.1.3 Packet Loss

Since neither IP Multicast nor RTP provides any reliability, packet loss have to handled either by a higher level protocol or by the applications.

MPEG-1 Layer III audio is particularly sensitive to packet loss because frames are not independent; a back-pointer in a frame points to the start of the data for this frame, and this often points to a position in the previous frame. If one frame is missing, several frames further on in the stream can be useless. Even low packet loss ratios such as 2-3% results in very bad audio quality at the receiver.

In this paper we present different methods of handling packet loss for MPEG-1 audio: Forward Error Correction, interleaving and a new packetization format. Together with Quality of Service techniques such as RSVP and receiver-based repairs such as error-concealment these are the main methods of handling packet loss. Combinations of these methods is also possible.

2.1.4 Relevant factors

There are several different criteria for evaluating the different methods, such as audio quality and latency. The main factors that was decided to be most important for the mIR application were:

Bandwidth usage Since we want to distribute the audio stream to many users, it is important that the bandwidth usage is as low as possible.

Audio quality Depends on both the packet loss and on the packetization format.

Reduction of packet loss Reduced packet loss means better audio quality

There are several other factors not considered, such as CPU usage, memory requirements and network latency. In for example a audio conferencing system the network latency must be kept to a minimum, but in a music broadcasting system such as mIR (see next section) a latency of a few seconds is often acceptable.

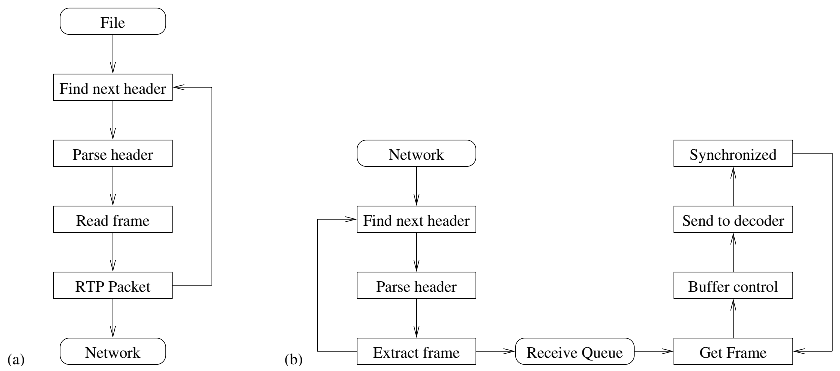

Figure 2.1: Send (a) and Receive (b) algorithms

2.2 Implementation

The multicast Interactive Radio - mIR - [24] applications are all implemented in the Java[3] programming language except for the MPEG Audio decoding which is done by an external decoder program such as Winamp[4]. or mpg123[5]. The two of the applications we consider in this paper are the sender and receiver applications. The mStar framework is used for network services such as RTP and multicast.

The sender application reads MPEG-1 Audio streams from storage and sends the data to a multicast group in a simple loop (see figure 2.1 (a)): 1. Find the next MPEG header. 2. Parse the MPEG header. 3. Read the rest of the MPEG frame. 4. Create and send an RTP packet to the multicast group. 5. Synchronize.

At the receiver end, two simple loops is used to receive and play the audio stream. The first loop is for receiving RTP packets from the network (see also figure 2.1 (b)): 1. Wait for an RTP packet from the network. 2. For each MPEG frame in the packet, parse the MPEG header and add the frame to the queue of incoming frames.

Another thread makes sure we forward MPEG frames to the decoder thread in the correct pace: 1. Get the next frame from the queue of incoming frames 2. If the queue is almost empty, sleep until the queue is reasonable full again 3. Forward the frame to the decoder process 4. Synchronize

2.2.1 Forward Error Correction

Forward Error Correction (FEC) support is implemented in the network layer (the mStar framework), and is based on RFC 2733, “An RTP Payload Format for Generic Forward Error Correction” [28]. It is a generic payload format for media encapsulated in RTP and not specific to MPEG Audio. FEC data is sent on to a separate multicast group so that non-FEC

[3] <URL:http://www.javasoft.com/>
[4] <URL:http://www.winamp.com/>
[5] <URL:http://www.mpg123.org/>>.

capable applications still can receive the media stream. It allows for recovery of both the RTP payload and critical header fields

It is engineered for FEC algorithms based on the exclusive-or (parity) operation. The payload format allows end systems to transmit using arbitrary block lengths and parity schemes. A version of this format that uses a Reed-Solomon based erasure code instead is also implemented.

No changes to the mIR sender or receiver applications where necessary, except requesting FEC traffic instead of best effort traffic from the network layer and specifying the multicast parameters for the FEC data stream.

Different FEC schemes can be loaded at start up time, using the dynamic class loading features the Java Platform provides. An simple API makes it easy to create new schemes without the need to recompile the complete application.

2.2.2 ADU Packetization Format

The standard for encapsulating MPEG-1 Audio in RTP is defined in RFC 2250 [18], and is very simple: a payload specific header are followed by one or more MPEG-1 Audio frames.

For Layer III this is far from optimal, since all data necessary to decode a frame can be spread over several packets. A loss of one packet can therefore force the receiver to discard several more frames.

The new packetization format is based on [11]; in this payload format the MPEG frames are rearranged, so that packet boundaries coincide with true MPEG Layer III "Application Data Units".

MPEG-Audio frames for a certain bit-rate and sampling frequency have a fixed length, but the actual encoded data for each frame can be smaller than this fixed length. The unused bits are called the bit reservoir and can be reused by future frames that require more data that can fit into the fixed length of one frame. By rearranging the data so each frame header is followed by all the data for the frame we get a stream of Application Data Units, ADUs [8]. These ADUs can be converted back to the original MPEG Audio stream.

If one ADU is lost, we can still use the data from the next ADU when we reconstruct a MPEG Audio stream by inserting "dummy" frames that serves as a container for the bit reservoir. Figure 2.2 shows the different packetization formats. Figure 2.2 (a) describes a normal MPEG Audio stream of frames, figure 2.2 (b) shows the stream after it have been converted to a stream of ADUs and figure 2.2 (c) shows how a dummy packet is inserted when reconstructing the stream of frames from a stream of ADUs in the case of packet loss.

The send and receive loops have to be modified to take advantage of the new packetization format. See figure 2.3 and 2.4.

The ADUs can also optionally be rearranged in an interleaving pattern. The purpose of interleaving is to reduce the effect of packet loss by distributing consecutive ADU frames over non-consecutive packets.

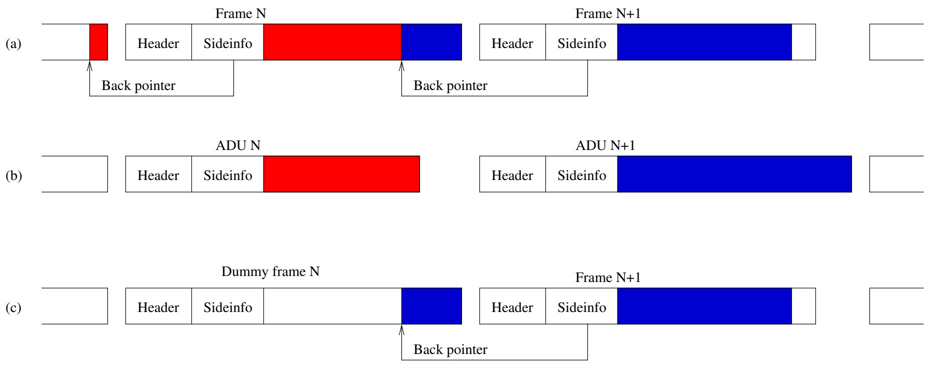

Figure 2.2: Mpeg Frames and ADUs. (a) Stream of MPEG Audio frames. (b) Stream of MPEG Audio ADUs. (c) Reconstruction of a stream of frames with packet loss

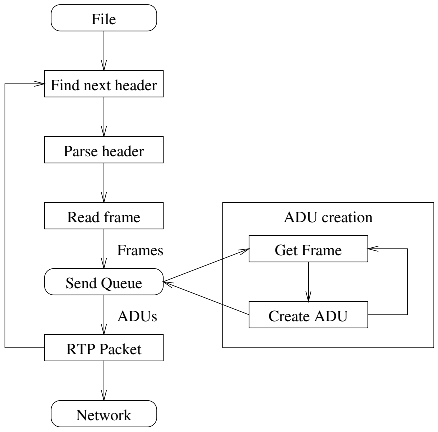

Figure 2.3: Send loop when using ADUs

2.3 Other methods

The main other methods for handling packet loss is reliable multicast and Quality of Service mechanisms.

Quality of Service mechanisms such as RSVP can allocate bandwidth for the audio stream and thus remove the problem of packet loss alltogether. The problems is that they are generally not available for use on the public Internet in any large scale.

Different reliable multicast protocols can also have the effect of removing all packet loss, although the delay and bandwidth usage can be highly unpredicatable depending on the network conditions, the number of clients, the specific protocol used, etc. A reliable multicast protocol [23] based on the Scalable Reliable Multicast, SRM [12] framework is implemented in mIR, but no measurements of the effectivness of this method have been done.

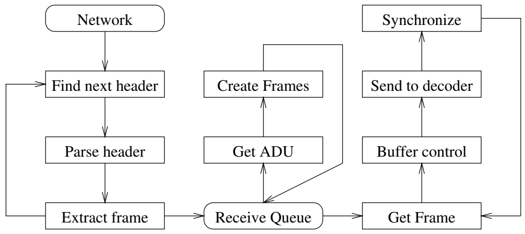

Figure 2.4: Receive loops when using ADUs

Trace	1	2	3	4
No FEC	1.56%	4.52%	11.2%	19.2%
RS(6, 3)	0.0%	0.08%	0.14%	1.1%
RS(12, 6)	0.0%	0.12%	0.8%	0.48%
RS(24, 12)	0.0%	0.0%	0.0%	0.08%
RS(15, 10)	0.0%	0.20%	0.35%	2.6%
RS(27, 18)	0.0%	0.17%	0.0%	1.5%
Parity Scheme 1	0.0%	0.20%	0.20%	1.1%
Parity Scheme 3	0.0%	0.40%	1.3%	4.2%

Table 2.1: Packet loss before and after forward error correction

2.4 Results

To test the different methods loss traces of real multicast traffic from [32] were used to simulate packet loss. Four traces with different amount of packet loss were selected: 1) WRNDec16/anhur 2) WRNDec18/alps 3) WRNDec11/float 4) WRNDec14/pax.

One audio file was processed through the sender application and the artificial loss was applied to the stream of RTP packets. These RTP packets were processed by the receiver and the decoded audio was digitally recorded. Data such as packet loss before and after forward error correction was also stored.

To test the effect of interleaving and the new packetization format subjective listening tests were the users rated each sample against a "good" sample (no packet loss) and a "bad" sample (trace 4, 19.4% packet loss) was performed. The test application and test methodology is based on [31]. Four different test parameters were used: 1) Standard packetization format from [18]) 2) New packetization format (ADU) without interleaving 3) New packetization format with a interleaving cycle of length 8 4) New packetization format with a interleaving cycle of length 32.

The initial results from listening tests (see figure 2.5) shows that the new packetization format increases the perceived quality substantially when packet loss increases. Interleaving does not have any large effect on the perceived quality. However, the initial results may not be statistically assured since too few (11) tests have been done so far.

Method	Overhead
RS(6, 3)	100%
RS(12, 6)	100%
RS(24, 12)	100%
RS(15, 10)	50%
RS(27, 18)	50%
Parity Scheme 1	100%
Parity Scheme 3	75%

Table 2.2: Extra overhead used by the forward error correcting codes

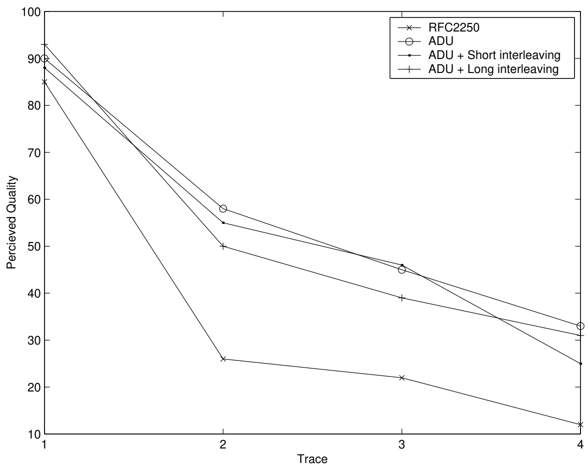

Figure 2.5: Perceived Quality vs. Packet Loss Trace.

Both the parity based and the Reed-Solomon based forward error correction schemes are efficient in reducing the packet loss ratio. Two parity based scheme was tested, scheme 1 and scheme 3 in [28]. See table 2.1 for the packet loss ratio for the four loss traces before and after forward error correction for the first 10000 packets. Long Reed-Solomon codes are the most efficient method, but even shorter codes provides sufficient protection against packet loss. Longer codes introduces more delay, so there is a tradeoff between efficiency and delay in the receiver. The extra overhead needed for the different schemes are listed in table 2.2.

2.5 Limitations and future work

The effect adding forward error correction have on the packet loss has not been considered. Since packet loss is strongly related to bandwidth usage the effect of adding forward error correction might actually increase the packet loss ratio, although the resulting packet loss after recovery will decrease.

The listening tests has not yet been done by enough users to make the results statistically significant.

The use of reliable multicast protocols have not been considered, although a implementation of this also exist. Different reliable multicast protocols varies a lot when it comes to scalability and bandwidth usage, which make them difficult evaluate.

2.6 Conclusions

In this paper we have describe several methods for improving the error resilience of MPEG Audio transmission in a lossy packet network.

The results show that the new packetization format is better than the standard packetization format for the same packet loss ratio. Forward error correction can be used to reduce packet loss ratio which further improve the audio quality. If the overhead of forward error correction is acceptable, the combination of FEC and ADU is the best option. Interleaving does not have a large effect on the perceived quality.

Part 3

Large scale distributed watermarking of multicast media through encryption

Large scale distributed watermarking of multicast media through encryption

Roland Parviainen, Peter Parnes Department of Computer Science/Centre for Distance-spanning Technology
Luleå University of Technology, 971 87 Luleå, Sweden
Roland.Parviainen@cdt.luth.se, Peter.Parnes@cdt.luth.se

February, 2001

Abstract

In this paper we describe a scheme in which each receiver of a multicast session receives a stream with a different, unique watermark, while still retaining the scalability of multicast. The watermarked streams can be used to trace those users who make unauthorized copies of a stream. The watermarking is enabled by encryption of two slightly different copies of the original stream with a large set of different keys.

3.1 Introduction

IP Multicast [9] 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.

Multicast is very well suited to use for large scale media distribution because of the scalability: each network link in the network only has to transport one copy of each packet regardless of the number of receivers. The drawback is that receivers do not have to be authenticated and can easily eavesdrop on the traffic without being detected. A unicast solution, where we send one copy of the stream to each user, is easier to protect but is infeasible for large scale transmissions to 100,000 to 1000,000 users and above.

Authentication and confidentiality can be solved with the use of encryption, but there is still a problem with malicious users retransmitting the media data unencrypted to other users. One way to detect whom the illegal copy originated from is *fingerprinting*, embedding unique information, a watermark, into each copy of the media that identifies the user receiving the copy. This information should be robust against any possible user manipulation, and in the remainder of this paper we will assume a robust watermarking scheme that can perform this fingerprinting exist. A fingerprinted stream might discourage illegal copying of the media, since the origin or the buyer of stream can be identified. This might be the only option for pure software solutions where tamper-resistant hardware is impossible.

The objectives of multicast and fingerprinting seem contradictory: multicast sends the same stream to everyone while to achieve the goals of fingerprinting every receiver should receive a different stream. In [4] Perkins, Brown and Crowcroft solve this problem by using active network elements that make sure all receivers get slightly different streams. In this paper we present a solution that does not suffer from the requirement of trusted active network elements. We propose a scheme were encryption is used to ensure different users receive fingerprinted streams. No trusted or active network elements are needed; all security is handled by the applications.

The remainder of this paper is structured as follows. In the next section we describe related work on multicast security. Then in section 3.3 we describe our approach in detail. In section 3.4 we describe possible attacks against the system. In section 3.5 an experimental implementation of this system is described. Sections 3.6 and 3.7 conclude this paper with limitations and conclusions.

3.2 Background

3.2.1 Multicast Security

Since IP multicast provides no authentication or confidentiality it is very easy to eavesdrop on, record and copy or retransmit a media stream completely anonymously. Basic encryption of multicast streams is not sufficient to protect important media streams since it is possible to retransmit either the content stream or the keys to untrusted users.

In [7] a 'virtual key' is marked instead of the plain-text. A certain minimum number of users need to collaborate to construct a key that works but identifies none of them. This does not prevent retransmission of media content and the bandwidth needed for the control messages may be too high.

Scalable content control schemes such as Nark [3] provides scalable authentication and encryption but need tamper-resistant smart-cards. Watermarking is not possible in todays limited smart-cards and have to be done off-card, but can be done using a system such as Chameleon [1].

Chameleon by Anderson and Manifacas is a similar scheme to ours, where a stream cipher is adapted to give slightly different output depending on a large unique key for each user. Our scheme can handle much more flexible watermarking algorithms; the two different watermarked packets do not have to have any common bits at all.

In the watercasting scheme [4] the source sends multiple subtly different copies of each packets and routers at the nodes in the multicast distribution tree discard packets, such that the stream delivered to each receiver is unique. This approach has several problems: support for this protocol in routers is needed which is probably hard to achieve, the routers have to be trusted, and the source must send d copies of each media packet, where d is the depth of the multicast tree.

3.2.2 Watermarking

A simple watermarking method is to change the least significant bits in for example an audio clip or an image. For an audio clip, we could put our embedded message into to least significant bit of some or all samples. These methods are easily broken. More advanced methods often use spread spectrum techniques or transforms such as Fourier and DCT to make the watermarks more robust.

A watermarking method that fulfills some requirements for the difficulty in removing it is called *robust*. Some examples of robustness requirements for audio recordings from IFPI, the International Federation for the Phonographic Industry [20] are:

- The sonic quality of the sound recording should not change
- The marking information should be recoverable after a wide range of filtering and processing operations, including two successive D/A and A/D conversions, MPEG compression, etc.
- There should be no other way to remove or alter the embedded information without sufficient degradation of the sound quality as to render it unusable

Similar requirements can be made for still images, video and other media types. A good introduction to the subject is [22].

3.3 Method description

The source sends two different copies of each media packet, each with a different watermark. Both copies are encrypted with two different, random encryption keys. The encrypted packets are then sent to all receivers using IP multicast. Any given receiver has access to the key of only one of the two encrypted packets of one media packet.

3.3.1 Transmission of packets

The source has access to k media packets: $P[1], P[2], \cdots, P[k]$ and an encryption algorithm E, such that $P = D(E(P,K),K)$. That is, $E(P,K)$ encrypts P with key k and $D(P,K)$ decrypts P. A watermarking algorithm W: $P_w = W(P,w)$, $w = U(P_w)$ and two watermarks, w_0 and w_1 are also needed. W embeds the watermark w in the cover object P, and U extracts the watermark from the marked object. A detection algorithm that detects if the watermark is still present can be used instead: $U(P_w, w) = B, B \in \{true, false\}$. The source needs $2k$ random encryption keys, $SK[1], SK[2], \cdots, SK[2k]$, to be able to transmit the media packets. A receiver r has access to k of these keys: either $RK_r[i] = SK[2i-1]$ or $RK_r[i] = SK[2i]$, $i = \{1, 2, \cdots, k\}$.

To transmit media packet i the source perform the following operations:

1. Read the media packet, $P[i]$
2. Create two watermarked packets, $V_0[i] = W(P[i], w_0)$ and $V_1[i] = W(P[i], w_1)$

3. Get two encryption keys: $SK[2i-1]$ and $SK[2i]$
4. Encrypt $V_0[i]$ and $V_1[i]$: $C_0[i] = E(V_0[i], SK[2i-1])$ and $C_1[i] = E(V_1[i], SK[2i])$
5. Transmit $C_0[i]$ and $C_1[i]$ together with i

At the receiver side, the client receives both packets and tries to decrypt them:

1. Receive two packets: $C_0[i]$ and $C_1[i]$
2. Get the decryption key for packet i: $RK_r[i]$
3. Try to decrypt both packets with key $RK_r[i]$
4. Only one packet will decrypt into a proper media packet: $V_{j_i}[i] = D(C_{j_i}[i], RK_r[i])$, $j_i \in \{0,1\}$
5. Decode and render $V_{j_i}[i]$

For each media packet the receiver will be able to decode exactly one of the watermarked packets. Which of the two packets is decided by the keys the source has assigned to the receiver.

3.3.2 Identity strings

If the keys a receiver have access to is unique among all receivers and known by the source, a unique identity string for that user can be defined: $id_r = B_r[1], B_r[2], \ldots, B_r[k]$, $B_r[i] \in \{0,1\}$.

This identity string can be derived by the source from both the keys given to the receiver and the stream the receiver decrypted. From the keys the source sent to the receiver:

$$B_r[i] = \begin{cases} 0, & \text{if } RK_r[i] = SK[2i-1] \\ 1, & \text{if } RK_r[i] = SK[2i] \end{cases}$$

From the decrypted stream for the user:

$$B_r[i] = \begin{cases} 0, & \text{if } U(V_{j_i}[i]) = w_0 \\ 1, & \text{if } U(V_{j_i}[i]) = w_1 \\ undefined, & \text{if neither } C_0[i] \text{ nor } C_1[i] \text{ was received or decrypted} \end{cases}$$

If the receiver do not receive all packets, due to for example packet loss or that the receiver tuned in late, the identity strings will not match completely. If n is large enough, the partial identity string will still be long enough to be unique among all receivers although some bits are undefined.

3.3.3 Bandwidth usage

Since 2 copies have to be sent for each media packet, the bandwidth usage is doubled both for the source and the receivers. This can be decreased by some optimizations.

At any given time, only one of the two watermarked packets is actually useful for a single receiver since the other packet can not be decrypted. If we send the two copies on different multicast groups the receivers can hop between the groups by joining and leaving them as the group the correct packet is transmitted on changes. In this approach we not only have to send the keys to each receiver but also which stream to receive; one extra bit for each key is needed. Unfortunately the join/leave latency for IP multicast is currently too large for this approach. Also, if more than one receiver is on the same network segment most of saving is lost.

As pointed out in [4] another optimization would be to only watermark 1 in every x packet, thus reducing the bandwidth to $(1+1/x)$ times the bandwidth of the original stream. Unfortunately a malicious user could remove these watermarked packets and retransmit the resulting degraded stream if x is large. We must therefore make sure that the degradation is large enough to discourage removal of the watermarked packets. One example of this is to only add watermarks to the I frames of an MPEG video stream or only watermark the last 10 minutes of a movie.

3.3.4 Key Distribution

The receiver keys can be treated as a long term key distributed by out-of-band means when the users registers, either as a downloadable file (protected by e.g. SSL/TLS [10]) or delivered to the user on a floppy or cdrom. All these solutions have problems when revocation of access is considered. The keys can also be continuously streamed to the users. The bandwidth per user needed for this is small, but it is still a serious scalability problem.

The amount of keys that each receiver needs depends on the required security (see section 3.4.1). The total size of the keys for one receiver is then $keys \cdot keysize$. Using 10000 keys and a key size of 128 bits thus requires 160 kbyte per user. The source only has to store a bitmask of length *keys* for each user and $2 \cdot keys \cdot keysize$ bytes of key material. A cryptographically secure random number generator can also generate the bitmasks instead to further reduce storage needs at the source.

3.3.5 Key size

It can be argued that attackers with sufficient funds to break, for example, 56 bit keys also have enough funds to get keys in other easier ways, for example using false identities while registering, and thus we do not need any stronger keys. On the other hand, getting access to computing power does not necessarily require monetary funds; a distributed attack is also possible.

It is not enough to break one of the keys since it only gives the attacker the option to change one of the watermarked packets in the stream. An attacker has to break a sufficient amount of keys to get enough packets to create an unidentifiable watermarked stream.

3.4 Attacks

We assume that it is not possible to either remove the watermark or break the encryption in reasonable time. We also assume that the attacker can not steal the non-watermarked stream from the source by breaking into the server.

If the encryption algorithm is broken an attacker can choose the final watermarked stream and make traitor tracing impossible, but if the encryption algorithm is chosen with care and with large enough key size and the keys are generated properly this can be avoided.

It is possible to attack another receiver's computer and steal the stream or keys from there, thus indicating someone else as the pirate. These kinds of attacks are out of scope for this paper.

Removal of the watermark might be possible if a robust watermarking algorithm is not chosen. The existence of such an algorithm is not considered in this paper.

Several users can collaborate and combine their streams to try to prevent watermark detection, either by choosing packets from different streams or by merging packets with for example bit voting. The two watermarks w_0 and w_1 and the watermarking algorithm should be chosen so that at least one of the watermarks is still present and recoverable after the bit voting.

Getting access to several streams and selecting packets from different streams is an easy and powerful attack. In the next section we analyze this attack further.

3.4.1 Traitor tracing

If p users collaborate, at least k/p of the original bits from one of the streams will always remain. We must make sure that this fraction is large enough to ensure a high probability of detection of the collaborators.

We assume we have one identy string for the illegal stream id_I and n identy strings for the legal watermarked streams: $id_{L[1]}, id_{L[2]}, \ldots, id_{L[n]}$. We want to identify the identy string of at least one of the streams that was used to create the illegal stream. We can calculate a measure on how good a watermarked stream match the illegal stream by adding the XOR value of bits from the identity strings:

$$M(L[i], I) = \sum_{j=1}^{k} B_{L[i]}[j] \oplus B_I[j]$$

We define the set S_{nc} as the values $M(L[i], I)$ where i is not one of the collaborators, that is $L[i]$ was not used to create the illegal stream. S_c is the values $M(L[i], I)$ where i one of the collaborators. The values in S_{nc} have a binomial distribution $X_{nc} = Bin\left(k, \frac{1}{2}\right)$, and the distribution of the values in S_c is $X_c = Bin\left(k\left(1 - \frac{1}{p}\right), \frac{1}{2}\right) + \frac{k}{p}$. These distributions can be approximated with a normal distribution,

$$X_{nc} = N\left(\frac{k}{2}, \sqrt{\frac{k}{4}}\right)$$

and

$$X_c = N\left(\frac{k}{2}\left(1+\frac{1}{p}\right), \sqrt{\frac{k}{4}\left(1-\frac{1}{p}\right)}\right).$$

As the number of collaborators, p, increases X_c converges towards X_{nc} as expected. If p is relatively small, the probability that all values in S_c are less than $E[X_c]$ is very high. For example, if $n = 100000, p = 10, k = 10000$ the distributions are $X_{nc} = N(5000, 50)$ and $X_c = N(5500, 47.43)$. The probability that at least one of the values in S_c is greater than $E[X_c]$ is $A = 1 - P(X_c < E[X_c])^{100000} = 1 - P(X_c < 5500)^{100000} = 0.76 \cdot 10^{-18}$ The probability that at least one of the values in S_{nc} is greater that $E[X_c]$ is $B = 1 - P(X_{nc} < E[X_c])^{10} = 1 - \left(\frac{1}{2}\right)^{10} = 0.9990$. That is, there is a high probability that the identy string with the highest value of $M(L[i], I)$ is one of the collaborators. For $p = 50$ on the other hand, we have virtually no chance to detect any of the collaborators.

3.5 Implementation

To test the feasibility of our scheme it was implemented in an existing Java application system[24] for audio transmission over multicast which uses the MPEG-1[14] audio compression standard. The system consists of a server application and a client application. The server read MPEG-1 audio data from disk and send it to a multicast address using RTP[27], the client receives this data and decodes it. The MPEG decoding is not done in Java but in native code.

Blowfish was chosen as the encryption algorithm, and was provided by the reference implementation of the Java^{TM} Cryptography Extension (JCE) 1.2.1 from Sun. Since the implementation is not supposed to be used in a production environment a very weak watermarking algorithm was chosen for simplicity: a few otherwise unused bits in the MPEG audio frame header are used for the watermark.

Prior to the transmission, the server generate $2n$ random keys and keys for p users, where the values n and p are given. These keys are stored in files until they are used. If the stream is longer than the value of n, the keys 'wrap around': the keys for packet i is $SK[1 + (2i - 1 \mod n)]$ and $SK[1 + (2i \mod n)]$.

When we enable the watermarking scheme in the server the CPU usage increases from 1.0% to 12%[†]. The large increase is because the server now has to encrypt every packet twice. For the client application the increase is from 6.7% to 18%. The bandwidth usage is the same for both a client and a server, independent of the number of users. As expected, the bandwidth[‡] usage increases by a factor of roughly two, from 133.3 kbyte/s to 270.4 kbyte/s. That the factor is 2.03 and not 2 is due to a small extra overhead for padding when doing encryption. The cost of key distribution is not included.

[†]Measured with perfmon.exe on a Pentium III 550 MHz computer running Windows NT4. The bit-rate of the media stream was 128 kbit/s and each key was 40 bits.

[‡]This includes RTP headers but not IP and UDP headers

3.6 Limitations

Our scheme relies on the existence of a robust watermarking scheme. No perfectly robust watermarking algorithm has been proposed so far, and it is not clear such a algorithm is possible. Without any optimizations, we need at least twice the bandwidth of the original media stream which might be too much for some applications. We have not considered the problem of revoking access for a receiver; with the current scheme this would require that new keys have to be distributed to all receivers again.

Although only the use of two watermarks are described in this paper we are not limited to this. Instead of sending two different packets for each media packet it is possible to send any number of copies, although this increases the bandwidth substantially. The watermarks w_0 and w_1 do not have be constant and can change at any time as long as $w_0 \neq w_1$ and the source keeps track of them.

3.7 Conclusion

We have described a scheme for scalable fingerprinting of multicasted media. Contrary to other proposed schemes no active network components or tamper-resistant smart-cards are needed. By increasing the bandwidth with a constant factor we can ensure that every receiver get a unique fingerprinted stream. The watermarks that make up the fingerprints are not fixed to a certain number of bits or format but can be of any format the watermarking algorithm requires for robustness.

Like other traitor tracing schemes based on watermarks, it has only limited collusion resistance. The most promising attack is to get hold of several streams and mix them together so the source can no longer be identified. The main countermeasure is to increase the number of watermarks in one stream.

References

[1] R.J Anderson and C. Manifavas. Chameleon – A New Kind of Stream Cipher. In *Fourth Workshop on Fast Software Encryption*, 1997.

[2] B. Briscoe. Marks: Zero side-effect multicast key management using arbitrarily revealed key sequences. In *First International Workshop on Networked Group Communication (NGC'99)*, 1999.

[3] B. Briscoe and I. Fairman. Nark: Receiver-based multicast non-repudiation and key management. In *ACM Conference on Electronic Commerce*, 1999.

[4] I. Brown, C. Perkins, and J. Crowcroft. Watercasting: Distributed watermarking of multicast media. In *Proceedings of the First International Workshop on Networked Group Communication*, 1999. Springer-Verlag Lecture Notes in Computer Science, 1736.

[5] R. Canetti, J. Garay, G. Itkis, Daniele Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and efficient constructions. In *IEEE Infocomm'99*, 1999.

[6] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure internet multicast using boolean function minimization technique. In *IEEE Infocomm'99*, 1999.

[7] B. Chor, A. Fiat, and M. Naor. Tracing Traitors. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270. Springer-Verlag, 1994.

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

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

[10] T. Dierks and C. Allen. The TLS protocol, 1999. IETF RFC2246.

[11] R. Finlayson. A more loss-tolerant RTP payload format for MP3 audio, 2001. IETF RFC3119.

[12] 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.

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

[14] 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.

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

[16] M. Handley and V. Jacobson. Sdp: Session description protocol, 1998. IETF RFC2327.

[17] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol, 2000. IETF RFC2974.

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

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

[20] London W1R 5PJ International Federation of the Phonographic Industry, 54 Regent Street. Request for proposals - Embedded signalling systems issue 1.0., June 1997.

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

[22] S. Katzenbeisser and F.A.P. Petitcolas, editors. *Information Hiding: Techniques for Steganography and Digital Watermarking*. Artech House, 2000.

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

[24] R. Parviainen. Multicast Interactive Radio. In *Proceedings of the Practical Application of Java*, pages 137–153, 1999.

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

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

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

[28] H. Schulzrinne and J. Rosenberg. An RTP payload format for generic forward error correction, 1999. IETF RFC2733.

[29] J. Schwenk, T. Martin, and R. Schaffelhofer. Tree-based multicast key agreement. In *Communications and Multimedia Security 01*, 2001.

[30] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8), 2000.

[31] K. Synnes, P. Parnes, and D. Schefström. Robust audio transport using maudio, 1999. Research Report 1999:04, Luleå University of Technology.

[32] M. Yajnik, J. Kurose, and D. Towsley. Packet Loss Correlation in the MBone Multicast Network. In *Proceedings of IEEE Global Internet*, 1996.

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