Distributing digital radio over an IP backbone

Master thesis in Control & Communication

By

Henrik Brunberg

Report no. LiTH-ISY-EX-3317-2002
Linköping 2002
Distributing digital radio over an IP backbone

Master thesis in Control & Communication
By
Henrik Brunberg

Report no. LiTH-ISY-EX-3317-2002

Advisors: Frida Gunnarsson and Jörgen Scott
Supervisor: Fredrik Gunnarsson

Linköping 2002-12-16
Distribuering av digital radio över IP backbone

Factum Electronics AB develops and manufactures equipment for Digital Audio Broadcasting (DAB). The increasing availability of IP networks has made the possibility to distribute the DAB signal, which is traditionally distributed over dedicated telecom links, over an IP backbone an interesting issue. This thesis investigates the possibilities to do this. Since distribution over IP is an unreliable service, it discusses techniques that can be used to eliminate the lack of reliability in IP. It also examines the advantages and drawbacks of using IP multicast, which would make it possible for many receivers to take part of the same signal.

During the work behind this thesis, a protocol was developed including features to make the distribution more reliable. A prototype application including a client and a server was constructed to test and evaluate the protocol. Implementation issues of the protocol and test results drawn from the application are described in this thesis.

IP, DAB, RTP, FEC, Multicast, QoS
Abstract

Factum Electronics AB develops and manufactures equipment for Digital Audio Broadcasting (DAB). The increasing availability of IP networks has made the possibility to distribute the DAB signal, which is traditionally distributed over dedicated telecom links, over an IP backbone an interesting issue. This thesis investigates the possibilities to do this. Since distribution over IP is an unreliable service, it discusses techniques that can be used to eliminate the lack of reliability in IP. It also examines the advantages and drawbacks of using IP multicast, which would make it possible for many receivers to take part of the same signal.

During the work behind this thesis, a protocol was developed including features to make the distribution more reliable. A prototype application including a client and a server was constructed to test and evaluate the protocol. Implementation issues of the protocol and test results drawn from the application are described in this thesis.
Concepts and abbreviations

ACK - Acknowledgement
Sent by a receiver to indicate that a packet has arrived.

ARQ – Automatic Repeat Request
General strategy for reliable transmission of packets over an unreliable link. The receiver sends an ACK to the sender when it has received a certain packet. If the sender doesn’t receive any ACK, the packet is sent again.

DAB – Digital Audio Broadcasting
A standard format for digital radio that is supposed to replace the traditional AM/FM radio.

Datagram
A message sent on a network that has no direct connection between the sender and the receiver.

Draft
A proposal for a standardization of some Internet function.

IP – Internet Protocol
The protocol used to distribute Internet traffic all over the world. It is said to be a protocol that can be used by nearly all kinds of networks.

Jitter
The variability of packet delays within the same packet stream.

Layer
An independent part of a protocol stack, see below, with a specified format for input and output.

Mpeg frame
Mpeg files consists of mpeg frames of various size depending of the encoding, each carrying some milliseconds of audio.

Mpeg header
Starts with some synchronization-bits to indicate the beginning of an mpeg frame, then follows information about the frame length, bit-rate, sampling frequency and other characteristics.
**Multicast**
A technique developed for IP that lets several receivers make use of the same signal. This is done by letting the receivers assign to a multicast group that has a certain IP address. The sender just sends its IP traffic to this IP address as it would in a one-to-one transmission.

**NAK – Negative Acknowledgement**
Message from the receiver to the sender telling it a certain packet wasn’t received and needs to be sent again.

**Octet**
A set of eight bits. Usually the same thing is meant by a byte, but the number of bits in a byte can vary.

**Packet**
What PDU’s, see below, are called in the IP layer. In this thesis however, the word is used as a general description of a set of data in a network.

**Payload**
The part of a packet that consists of the information we want to distribute.

**PDU – Protocol Data Unit**
When a set of data is transported down the protocol stack, it is modified and might be split up in smaller parts or put together in bigger parts. These are called different things in different layers, but the general concept is PDU.

**Protocol stack**
In networks, functionality for transmitting information is split up in different layers. At the sender, the upper layer pushes its data down to the underlying layer, which adds some more information to the data and pushes it to the next layer. When the information reaches the bottom layer, the information is actually transmitted over the network. At the receiver, the information travels the other way up the receivers protocol stack.

**QoS – Quality of Service**
Methods used in the network layer to ensure that certain traffic is guaranteed a certain bandwidth and maximum delay.

**RFC – Request For Comments**
A document describing some Internet function. If accepted, the document may become an Internet standard.

**RTT – Roundtrip Time**
The time it takes to send data to a node and get a response back from it.

**XOR – eXclusive OR**
A logical operation that takes two binary numbers. If one of them is one and the other one zero, the result is one, otherwise the result is zero.
Contents

Concepts and abbreviations .......................................................................................................... 9

Contents ........................................................................................................................................ 11

1. Introduction .............................................................................................................................. 13
   1.1. Factum DAB system ............................................................................................................ 13
   1.2. DAB over IP ...................................................................................................................... 15
   1.3. IP over DAB ...................................................................................................................... 16
   1.4. Reading guidelines .......................................................................................................... 16

2. Distribution with IP .................................................................................................................. 17
   2.1. Multicast ......................................................................................................................... 17
       2.1.1. How multicast works ............................................................................................... 18
       2.1.2. Multicast obstacles ................................................................................................. 19
   2.2. TCP – Transport Control Protocol .................................................................................. 19
   2.3. UDP – User Datagram Protocol ..................................................................................... 20
   2.4. RTP .................................................................................................................................. 20

3. Reliable distribution .................................................................................................................. 22
   3.1. Quality of Service .......................................................................................................... 22
   3.2. Retransmission ............................................................................................................... 24
   3.3. Forward Error Correction .............................................................................................. 25
       3.3.1. Using FEC to eliminate packet loss ........................................................................ 26
       3.3.2. XOR .......................................................................................................................... 28
       3.3.3. Reed Solomon ......................................................................................................... 28
   3.4. Interleaving ..................................................................................................................... 29
   3.5. Error hiding .................................................................................................................... 30
   3.6. Buffering strategies ........................................................................................................ 31
   3.7. The solution of choice .................................................................................................... 32
       3.7.1. Using RTP .............................................................................................................. 33
       3.7.2. Integrating the FEC methods with RTP ................................................................. 34

4. Prototype .................................................................................................................................. 36
   4.1. Sender ............................................................................................................................. 36
   4.2. Receiver .......................................................................................................................... 38

5. Testing ..................................................................................................................................... 42
   5.1. Packet loss recovery ...................................................................................................... 42
   5.2. Hardware load ............................................................................................................... 43
1. Introduction

Factum Electronics has been developing and marketing digital audio equipment since 1988 and offers a wide range of standard systems components and custom specified solutions for signal encoding, decoding and processing. The DAB system traditionally distributes its signal via dedicated telecom links. With the growth of the Internet however, IP networks are getting more and more common, and it would be an advantage for operators to be able to use their IP backbones to distribute the DAB signal. The main purpose of this thesis is to explore the possibilities to do this. The evaluation should render in a prototype that makes it possible to test the results on an IP network. The network situation shown in Figure 1.1 might be the most typical example where such an application could be used, but the prototype should be general enough to be usable in an arbitrary IP network.

1.1. Factum DAB system

The Digital Audio Broadcasting system was developed by the Eureka 147 Project, an international consortium of broadcasters, network operators, consumer electronic industries and research institutes. These parties joined to develop the DAB standard, which has already been implemented in many countries worldwide.

![Figure 1.1 – Typical network situation](image-url)
Introduction

DAB is broadcast on terrestrial networks, with prospects for Satellite broadcasting in the future. CD-like quality radio programs are received even in the car without any annoying interference and signal distortion. DAB radio can carry not only audio, but also text, graphics, data and even videos.

A DAB distribution network (Figure 1.2) contains the following main parts/actors:

Content Provider
Radio studio where the programs and data services are produced.

Service Provider System (SPS)
The system responsible for encoding audio, handling data, creating control information etc. and also delivering a DAB service multiplex to the next level in the network.

Ensemble Provider System (EPS)
The system responsible for managing all inputs and creating the final DAB ensemble multiplex for channel encoding and transmission.

Channel encoding and transmission
The system responsible for encoding the ensemble for the actual transmitters and frequency.

Receiver
The DAB radio or other equipment able to receive DAB.

Factum specialises in Service Provider and Ensemble Provider Systems based on the DAB standard.

The systems are intended for various companies involved in radio broadcasting, such as:

- National public or private broadcasting companies.
- Local public or private broadcasting companies.
- Ensemble or network providers.

It gives listeners interference-free reception of high-quality sound, easy-to-use radios, and the potential for a wider listening choice through many additional stations and services.

![DAB Systems by Factum](image)
Introduction

Eureka DAB is a reliable multi-service digital broadcasting system for reception by mobile, portable and fixed receivers with a simple, non-directional antenna. It can be operated at any frequency from 30 MHz to 3 GHz for mobile reception (higher for fixed reception) and may be used on terrestrial, satellite, hybrid (satellite with complementary terrestrial) and cable broadcast networks. In addition to supporting audio programs with a wide range of sound coding rates and qualities, it also has a flexible, general purpose digital multiplex which can carry a number of services, including audio-program associated data and independent data services.

1.2. DAB over IP

Because of the increasing availability of IP networks, Factum would like to provide a possibility for their customers to use IP technology to distribute the DAB signal between the audio encoder and the multiplexer shown in Figure 1.1. corresponding to the arrows between the SPS:es and the EPS in Figure 1.2. These nodes are hereafter considered as sender and receiver respectively. The standard used today is built on using telecom links dedicated only for digital radio. If a good technique was developed for distributing digital radio over IP it could also be used in other places in the chain of distribution. In some places for example, the signal has to go between two elements that are physically in the same room, and if the signal were converted to go over IP, expensive communication cards could be exchanged for common Ethernet cards.

Since digital radio is a real-time medium and IP is a protocol not guaranteeing that all information reach its destination complete, in the right order, or even at all, this has to be taken care of by protocols on a higher level than IP. One can consider using retransmission (ARQ) or error correcting codes to create guarantees for the information reaching its destination as expected. However, ARQ, error correcting codes and such mechanisms can never fully cover for the lack of reliability in an IP network. If a network contains bottlenecks not letting enough information through, the result can be the main part of all IP packets sent disappearing, making reliability methods used in higher layers useless. Therefore, one must consider using techniques as resource allocation and quality of service (QoS) to guarantee that traffic is given the bandwidth needed.

Everything that is done to make the signal reach its destination in a complete condition is somewhat time demanding. The techniques used will make it necessary to use buffers both at the receiver and at the sender. Since digital radio is a real-time medium being compared to the AM/FM-radio it is sensitive to delay. Live interviews and such events common in today’s radio for instance, would be problematic with the sound being delayed a number of seconds, since the radio transmission is often used as a feedback channel by the reporter. The delay is one of the most important aspects that customers take into consideration when investing in DAB equipment. Therefore, it is necessary to in every step of the process weigh the advantages of reliable transmission against the disadvantages of increased delay due to the need for buffering of the signal.

The most general use of the results expected from this thesis is the case when there is an IP backbone with several audio encoders and multiplexers connected to it (Figure 1). All multiplexers should then be able to receive signals from optional audio encoders for broadcasting in the air. This is where the technique for IP multicasting gets into the picture. It is far more effective for a sender to send a signal to all receivers once, than to establish single
Introduction

connections with all receivers and thereby send the same information several times. Multicasting would also make it easy for receivers to start receiving a signal from an arbitrary sender, since all they have to do is to assign themselves to a multicast group; the sender doesn’t even have to know about it. However, the possibility for traditional communication with unicast IP traffic must be kept open, since some customers might be unwilling to configure their IP networks for multicast traffic.

1.3. IP over DAB

The evolution of Internet has created a new set of solutions to build streamed services and e-commerce applications based on IP. These systems may be directly applied to the IP over DAB technology. IP over DAB provides several advantages, such as high bandwidth, high penetration, high performance encoding and low cost due to point to multipoint transmission, thereby combining the best of DAB and wireless Internet radio.

An interesting perspective of the work behind this thesis would be to see if the techniques developed for solving the main problem, to distribute DAB over IP, could be used to improve the IP over DAB technology. Methods developed to perform reliable transmission of IP traffic could be used to improve the IP over DAB traffic and provide possibilities to lower the reliability demands of the DAB signal, since lack of reliability would be taken care of at higher levels.

1.4. Reading guidelines

To make the reading of this thesis more efficient, a brief description of each chapter is gathered below.

1. Introduction
Discusses the purpose of this thesis and provides background information of the DAB system.

2. Distribution with IP
Description of the IP protocol, its possibilities and drawbacks. The transport protocols available for IP; TCP and UDP, and the Real-time Transport Protocol, RTP, are discussed. This chapter also provides a presentation of the multicast technology.

3. Reliable distribution
Methods for making the transmission over IP reliable are discussed. In the end of the chapter, a solution for an appropriate protocol for the task is presented.

4. Prototype
A prototype was constructed to test the theories presented in this thesis. Its functionality and some implementation issues are discussed in this chapter.

5. Testing
Tests, considering reliability improvement and hardware load, are presented in this chapter.

6. Results
Results derived from the background studies of this thesis, implementation issues of the prototype and the test results are discussed.
2. Distribution with IP

The IP protocol has become popular because of its easiness and its scalability. It has a connectionless model of data delivery that is sometimes called best effort, because although IP makes every effort to deliver datagrams, it makes no guarantees. It is therefore sometimes called an unreliable service. Every datagram carries enough information to let the network forward the packet to its correct destination, and an unreliable service like this is about the simplest service you could ask for from a network. The ability of IP to “run over anything” is frequently cited as one of its most important characteristics; some of the technologies over which IP runs today didn’t even exist when IP was invented. The unreliability of IP networks doesn’t only mean that packets can get lost, they can also be delivered out of order or after some arbitrary long delay and sometimes one packet can be delivered more than once. Of course, interferences in the network could also make the IP packets contain erroneous information, but this is pretty unusual in a common network, since mechanisms in lower layers often sees to it that erroneous information gets silently dropped. These facts have to be considered by protocols and applications that run above IP.

On top of IP there has to be a transport protocol. A transport protocol is sometimes called an end-to-end protocol and gives processes ability to communicate directly with each other. It also often provides mechanisms to compensate for IP’s lack of reliability. The port number is an important issue of the transport protocol, since this is what separates the communication between different applications from each other. Every application uses its own port number and this way the host can tell which process should receive which packet. The IP protocol is often mentioned together with TCP (TCP/IP), since this is the most common transport protocol used on the Internet. However IP leaves two choices when it comes to what transport protocol to use; TCP or UDP, who are further discussed in Sec. 2.2 and Sec. 2.3 respectively.

2.1. Multicast

The advantage of traditional unicast (one-to-one) communication is that the sender can establish a direct connection with the receiver and make it easy to use retransmission techniques to achieve reliable communication. The sender and receiver can then have an advanced dialog about what information to send and when to send it. The disadvantage is that every receiver that wants to take part of the communication needs its own unique connection to the sender. This leads to the same information being sent one time for each receiver wanting it, which of course means wasted bandwidth. Multicast deals with this problem and lets multiple receivers share the same transmission from a sender who only has to send the
Distribution with IP

information once. Another advantage of multicast is the easiness with which a receiver can start receiving information from a sender. The sender doesn’t even have to know which receivers take part of its traffic.

2.1.1. How multicast works
Multicast makes it possible for multiple receivers to take part of the same information without the sender having to send it more than once. This is done by so called multicast groups. The sender distributes its information to the multicast group distinguished by an IP multicast address. The multicast address is chosen from a group of IP addresses that are dedicated for multicast use in the address range 224.0.0.0 to 239.255.255.255. For the receiver to take part of the information sent to a multicast group, it has to join that group. This is done by using a protocol called Internet Group Management Protocol (IGMP), which is used to handle the administration necessary to administrate multicast communication. Receivers use this to notify a router on their local network of their desire to receive packets sent to a certain multicast address. Both the sender and the receiver must support this protocol in their so-called TCP/IP stacks to be able to send and receive multicast messages.

The way that multicast group memberships and routing of multicast messages is handled by a network is somewhat beyond the scope of this thesis, since it is pretty much up to the network provider to configure the network to work for IP multicasting. However, a brief presentation of the principles of the technique follows.

Hosts, wanting to take part in a multicast transmission, join a host group. A multicast datagram is delivered to all members of its destination host group with the same best-effort reliability as regular unicast IP datagrams, i.e., the datagram is not guaranteed to arrive intact at all members of the destination group or in the same order relative to other datagrams. The membership of a host group is dynamic; hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time.

Two new operations have to be introduced to a host that wants to be able to receive multicast datagrams:

- JoinHostGroup
- LeaveHostGroup

These operations, as their names imply, exist to administrate the hosts group memberships. The IP module on a host must be extended to maintain a list of host group memberships to be able to support the reception of multicast IP datagrams.

The IGMP is used by IP hosts to report their host group memberships to any immediately neighbouring multicast routers. Multicast routers send Host Membership Query messages to find out which host groups have members on their attached local networks. Hosts respond to such a query by generating Host Membership Reports, reporting each host group to which they belong to the network interface from which the query was received. This procedure is carried out so that the multicast routers knows what IP multicast addresses it should listen to. If for instance all neighbouring hosts should all have ended their membership in a group, the router no longer needs to receive the messages addressed for this group. When a host joins a new group, it should immediately transmit a report for that group, rather than waiting for a
query from the router, to ensure that the router will be listening for messages destined for that group.

2.1.2. Multicast obstacles
To enable a backbone for IP multicast, some conditions of the network have to be fulfilled. First of all, the hardware has to be able to handle multicast traffic. Today, the main part of all major manufacturers of network elements support IP multicast, making sure their products have this capability. However, one has to make sure that the equipment in the backbone is up to date; if an old switch not supporting IP multicast should reside somewhere in the network, it could make the entire network unsuitable for our purpose. Furthermore, some corporations might not want to configure their backbone to work with IP multicast because of the fear that multicast traffic would interfere with the regular traffic already there, and therefore solutions using multicast can’t be expected to be suitable for every network.

2.2. TCP – Transport Control Protocol
It is common that real-time applications on the Internet use the TCP protocol to distribute media like audio or video. Therefore it would be meaningful to investigate the possibilities to use such a solution for our application.

TCP wraps its information in segments containing both data and control information. It establishes something called virtual circuits between hosts, which behave as if there existed a direct connection between the communicating computers. Each segment is somewhat dependent of the segment before and the segment after, which means that a lost segment could do more harm than just losing the information in that particular segment. Furthermore, the first and last messages in a TCP session demands special care, since this is where the virtual circuit is established and ended.

TCP uses techniques to measure the performance of a connection and adjusts the size and sending speed of the segments according to these measurements. This means that one has no control over at what pace TCP will send the information one wants to distribute, which of course is a serious problem in our case, since digital radio demands a certain transfer speed that has to be reached to build a proper signal at the receiver.

Bandwidth
Retransmission techniques are used by TCP to guarantee delivery of information. This contributes to the fact that TCP isn’t bandwidth effective (Sec. 3.2). It also sends a lot of overhead information to administrate the virtual connections, causing further waste of bandwidth.

Delay
TCP is error proof in the sense that one can be sure that information will arrive. However, nothing is said about when it will arrive. The sender always has to listen for messages from the receiver acknowledging that segments have arrived and sometimes wait for these acknowledgements before permitted to send more segments. This means high risk for delay of the signal since one always has to deal with some percentage of packet loss from the underlying network. In real-time applications this delay problem is often dealt with by having a time buffer covering for variations in speed of delivery and guaranteeing that the data
Distribution with IP

supposed to be presented at a given moment always has arrived. This however, causes a constant delay of the signal which is not suitable for our application, as discussed in Sec. 1.2.

Multicast
TCP is designed to be used for host-to-host communication only and is not compatible with IP multicast.

Conclusion
Since digital radio has demands of low delay and a predictable bit-rate, it would be a bad idea to use TCP for our purpose. Furthermore, our desire to be able to take advantage of IP multicast would be impossible to fulfil using TCP.

2.3. UDP – User Datagram Protocol

UDP, like its underlying protocol (IP), lets issues about reliability be dealt with by the layers above it. It doesn’t provide any support for error correction, retransmission or control of the packets delivery order. What it does provide is for the applications to communicate directly with each other, since its header (Figure 2.1) contains a field for port number. It also silently drops packets that have bit-errors using a checksum detecting such failures.

<table>
<thead>
<tr>
<th>Port number at the source</th>
<th>Port number at the destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP length</td>
<td>UDP checksum</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.1 - UDP header

Since TCP is out of the question to use for our application, we are forced to use UDP, which really doesn’t provide any functionality. To ensure in the highest degree possible that the audio signal arrives at the receiver without interference, with the smallest possible delay, we need a protocol above UDP in the protocol stack. A standardized protocol developed for real-time applications is the Real-time Transport Protocol, RTP. It also is well suited for IP multicast.

2.4. RTP

The Real-time Transport Protocol is a mix between a transport protocol and an application protocol, since it has all the properties of a transport protocol but is often implemented as part of the application. It provides functions suitable for transmitting real-time data like audio or video over a multicast or unicast link. No standardized functions for reliable transmissions are provided within the protocol, but it is well suited for adding such functionality. It also has a control protocol, RTCP (Real-time Transport Control Protocol), which supervises the traffic. RTP and RTCP are developed to work independent of the layer beneath. The most common application though, is to use RTP on top of UDP.

The main functions that RTP provides are identification of contents, sequence numbering and timestamps. Its easiness makes it appropriate to use with IP multicasting, even if one wants to supervise the traffic with RTCP and thus get communication in both directions. The sequence numbering and its compatibility with IP multicasting are the main reasons for RTP being interesting for our application. RTP also makes it possible to pad packets, which means adding an arbitrary number of zeroes at the end of a packet and making it possible to give all packets the same size. This is necessary when using forward error correction, which is
Distribution with IP

explained in Sec. 3.3. There is also lots of information available on how to enhance RTP with reliability functions such as forward error correction. RTP has become a de-facto standard for real-time communication, and sticking with standards is always good for compatibility issues that might show up in the future. The RTP header is shown in Figure 2.2.

<table>
<thead>
<tr>
<th>V=2</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>Sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Synchronization source (SSRC) identifier</td>
<td>Contributing source (CSRC) identifiers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2 - RTP header

The fields in the RTP header have the following meaning:

V – RTP version
Set to 2 if the specification in [3] is used.

P – padding
If set, the packet contains one or more payload octets, which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored.

X – eXtension
If set, the header is extended with more information.

CC – CSRC Count
The number of CSRC identifiers that follow the header.

M – Marker
Bit used to let certain packets in a stream be marked by reasons optional to the application.

Sequence number
Incremented by one for each RTP packet sent. May be used by the receiver to detect packet loss and to restore packet order. The initial value of a stream should be chosen randomly to be unpredictable. This is to make attacks on encrypted packets more difficult.

Timestamp
Reflects the sampling instant of the first octet in the RTP data packet. The initial value in the timestamp field should be chosen randomly by the same reason as for the sequence number.

SSRC
Identifies the synchronization source. This number can be used to separate different RTP streams if the IP address and port number are the same. The number is chosen randomly by the same reason as for the sequence number.

CSRC
List of 0 to 15 items. Identifies contributing sources for the payload contained in the current packet. Used by mixers who insert SSRC values of contributing sources in these fields.
3. Reliable distribution

The problems one has to deal with when using a solution with UDP on top of IP is:

- Packet loss
- Packets being delayed and arriving out of order
- Packets arriving more than once

The most serious of these problems is the packet loss problem, since the other ones can be solved by using simple buffering strategies. There are great differences between different networks concerning the probabilities of packet loss. A typical percentage of packet loss on an arbitrary network though is said to be about 2-5% [5]. The probability of two packets after each other getting lost is typically rather low, even though it is more probable that a packet after a lost packet gets lost than for an arbitrary packet to get lost. Therefore burst errors, meaning that several packets in a row get lost, is a rather unusual but existing phenomena.

The goal for the layers above the IP layer will be to eliminate the consequences of one or a smaller amount of packets getting lost, while the lower layers should be set up to minimize the risk for packet loss. The numbers presented above are highly dependent of how the network is built, the risks are of course reduced drastically if the network is dedicated mainly for the purpose of distributing digital radio. However the methods developed in the layers above IP should be general enough to be able to deal with various rates of packet loss and other IP-related problems.

3.1. Quality of Service

Quality of Service, QoS, deals with issues of providing a better service for certain types of traffic in a network. The aim is to give priority to this traffic in terms of bandwidth, controlled jitter, latency and probabilities of packet loss. The main issue for solving the problems with our application is to know what kind of service we will get from the network. Our traffic doesn’t necessarily have to be treated better than other traffic, but we need to know that we will be able to use a certain bandwidth and have approximate numbers on the probability for packet loss and delay. Therefore, some kind of quality of service technique would be appropriate for the underlying network to provide. Our distribution could not go across some arbitrary part of the Internet, where we have no control of bandwidth availability or transmission reliability. It should also be mentioned that if a network is completely dedicated for our cause, to distribute digital radio, which is somewhat suggested in Figure 1.1, issues of
Reliable distribution

QoS are unimportant. Then all traffic have the same priority and we can continue to put load on the network until we reach the limit for what is possible.

The idea with QoS is to provide a service model able to meet the demands from different kinds of applications, prioritizing certain traffic for example when queues appear, and thus achieve a lower risk of getting dropped if for example a queuing buffer in a router should get filled. QoS can be applied to an entire network, or just on the network elements one is able to control.

There are two main approaches to QoS; Integrated Services and Differentiated Services.

Integrated Services – IntServ

Integrated services consists of specifications of a number of service classes designed to meet the goals of applications that needs extra care in terms of network performance. It is closely connected to the RSVP (Resource reservation protocol), which is used to reserve resources for these service classes. There are two main types of service classes in the Integrated Services:

- Guaranteed service – the network should guarantee that the maximum delay that any packet will experience has some kind of specified value.
- Controlled load – aims to emulate a lightly loaded network for those applications that request the service, even though the network might in reality be heavily loaded.

To make a network provide these services, we need to tell it not only what we want, a maximum delay of 100 ms for instance, but also how much data we will inject into it. The set of network information that we provide to the network is referred to as a flowspec. A flow is a name for a certain traffic that travels from a certain source address on a certain port to a certain destination address on a certain port, and flowspecs are used to inform the network about the needs of these flows. For the network to be able to accommodate a new request for service, it needs some kind of admission control. The admission control looks at the flows demands to see if the desired service can be provided without causing any previously admitted flow to receive worse service than requested. Thus, the flow is simply either admitted or denied. To make this decision, the network has to use algorithms and heuristic methods.

Connectionless networks rely on little or no state being stored in the network itself, making it possible for links to go up and down while end-to-end connectivity is still remained. The RSVP tries to maintain the robustness of connectionless networks by using the idea of soft state in the routers. Soft state in contrast with hard state, found in connection-oriented networks, does not need to be explicitly deleted when it is no longer needed. Instead, it times out after some period of time if it is not periodically refreshed. RSVP has a receiver-oriented approach for resource reservation. Each receiver periodically sends refresh messages to keep the soft state in place. For a receiver to make a reservation, it needs to know what traffic the sender will transmit and what path the packets will follow, so that it can make reservations along this path in the network. Thus, setting up and maintaining a resource reservation takes a fair amount of administrative traffic to be sent across the network. Once the actual transmission has started, the network has to deal with the classifying of packets, to be able to handle different flows correctly, and to manage the packets in the queues so that they receive the service that has been requested.

23
Reliable distribution

As a conclusion, it could be said that IntServ needs a lot of work under the hood of the network, keeping the states of different flows and periodically sending refresh messages.

**Differentiated Services - DiffServ**

Differentiated Services is a more straightforward way of QoS compared to Integrated Services. Instead of using a protocol like RSVP to tell all the routers that some flow is sending packets that should be prioritized, DiffServ lets the packets identify themselves. The packets are identified with different levels of priority by setting predefined values in the TOS field in the IP header. TOS stands for Type Of Service and is unused in common IP traffic. The behaviours of the routers when receiving a prioritized packet are called “per-hop behaviours”, indicating that they define the behaviour of individual routers rather than end-to-end services. Depending of the value in the TOS field, the routers could then for example be told to forward a packet with minimal delay and loss. Packets that would be considered for getting a lower degree of priority in our case are simply packets not needing a real-time treatment, such as packets belonging to e-mails, web-pages and so on. For the router to be able to forward a packet with a minimal delay and loss probability, it has to be sure that the arrival rate of such packets is lower than the rotters capacity. This has to be accomplished by just allowing a certain amount of this kind of prioritized packets. If we were to distribute our traffic on a network configured like this, we could consider it our own, provided our traffic had the highest priority, since we would never be interfered by other types of traffic. DiffServ leaves lots of configuration issues to the network administrator, and assuming we are able to make these configurations, using our own network or buying ability to prioritize our traffic from a network provider, DiffServ would be a good way to ensure a low probability of packet loss and a minimal delay for our traffic.

Comparing the methods discussed above, DiffServ would be a good choice for our application, since it is a straightforward way of achieving QoS and has small demands of underlying network activity. However, any of these methods could be used and the choice is left to the implementer of the system, having access to the facts and possibilities of the current network.

**3.2. Retransmission**

One way of preventing the loss of packets is that the receiver tells the sender either when it misses a packet or every time it receives a packet. With this information, the sender knows if a certain packet got through or if it should try to send it again. ARQ is a general method for ensuring that all information gets through in a network. The idea is that when the receiver receives a message, it sends an ACK to the sender. If the sender gets an ACK corresponding to a certain message, it knows the message got through and doesn’t have to send it again. Otherwise, the sender waits for some time and tries to send the message again. A variation of the technique is for the receiver to send a NAK message to the sender every time a packet is missing, causing the sender to send the packet again.

Stop-and-wait is the simplest form of ARQ. This algorithm doesn’t allow the sender to send a new packet until it gets an ACK for the previous packet and is not bandwidth-effective, since the sender can only send one packet per RTT (Roundtrip time); there can only be one packet in the link. There are also ARQ techniques far more sophisticated than the stop-and-wait algorithm that are more bandwidth-effective, though more complicated to implement.
Reliable distribution

Retransmission is a good method when you want to know for sure that information transfers from one node to another. For the purpose of this work however, this method has a few drawbacks that makes it unsuitable to use. First, retransmission techniques are not designed for real-time applications. If the RTT is high, a lost packet could cause severe latency in the communication, and not knowing how many times a packet needs to be sent to arrive at the receiver, the delay gets very unpredictable. Perhaps this problem could be eliminated by having a time buffer covering for variations in delay, making sure that we always have the information we need, but such a buffer would have to be of such size that the solution wouldn’t live up to the real-time demands of digital radio. Secondly, retransmission methods aren’t suitable for IP multicast. If a network has many nodes, it is likely that one of them doesn’t receive a packet. This means the sender has to multicast that packet to all nodes again, causing the nodes to have to deal with multiple delivery of the same information. The sending of ACK messages from all receivers to the sender also would cause great load on the networks bandwidth resources, while using the NAK method would force the sender to deal with multiple NAK messages for the same packet.

There are in fact methods for using retransmission in a multicast environment in a smart way, suggesting for example nodes that doesn’t receive a packet request this packet from the closest node that received it. Such methods though makes it a lot more difficult to come up with a general solution for an arbitrary network and requires a lot of knowledge about the network where they should be implemented.

The drawbacks presented above makes it easy to decide that retransmission methods shouldn’t be the choice for reliable transmission in our case, since we want a general solution well suited for real-time transfer, independent of the network structure and able to benefit from the advantages of IP multicast.

3.3. Forward Error Correction

Forward error correction, FEC, is a technique often used in one-way communication, since it doesn’t require feedback from the receiver. The idea is that redundant information, calculated with mathematic tools, is sent along with the main information. If information is lost during transmission, the redundant information can be used to recreate the missing information. Before FEC is implemented, one has to think about the impact this will have on a network. Since FEC means more load on the network, if the network is sensitive to high traffic, it could mean increased probability for packet loss. It could also be time demanding due to time-consuming calculations, making it impossible to use for real-time applications.

There are two main methods for generating error-correcting codes. One is payload independent, while the other requires knowledge of the payload characteristics to optimize the error-correcting process. The payload-dependent method provides possibilities to make more effective FEC codes, but requires knowledge about how the information is structured, like for instance what compression method is used. This makes the calculations more difficult to implement than the general solution since the mathematics get more complicated and the only thing gained is less overhead, since less redundant information is needed to recreate the missing information. Since we want to keep our solution as simple and general as possible, making it possible to use any audio format in the payload, the general method is the best choice. The assumed format of the payload our application should deal with is the DAB format, which uses Mpeg layer II, but we don’t know if this format could change in the future, and therefore it’s better to stick with a general solution.

25
Most FEC methods are configurable, making it possible for the user to decide how much redundant data to create for a certain amount of information. This is specified through the n- and k-values. The n-value defines how many symbols exist when information and redundant data is added together, while the k-value defines how many of the n symbols are information symbols. Hence, $n - k$ represents the number of redundant symbols. The values of these symbols are often denoted as $(n, k)$ without any further explanation.

There are lots of different error-correcting codes available that are used in communication and storing applications. To understand how they work, some advanced mathematical knowledge about concepts like groups, rings and finite fields is required. For the continuation of this investigation it was decided to choose two error-correcting codes for implementation, one very simple and one more advanced and effective. The two methods chosen were the XOR-method and the Reed Solomon method, who are further discussed in Sec. 3.3.2 and Sec. 3.3.3 respectively.

**3.3.1. Using FEC to eliminate packet loss**

When UDP is used, no concern has to be taken for packets containing erroneous information, since these packets are silently dropped due to the UDP checksum. What FEC used with UDP has to deal with is instead the problem of packets not reaching their destination.

All FEC methods are originally designed to deal with bit errors or erasures in a context of lost bytes. That is, $n$ bytes are sent of which $k$ are information bytes and $n - k$ are redundant bytes. The advantage of packet loss compared to usual bit-errors is that using the sequence number functions of RTP, we always know where the error is, thus suffering from erasures instead of errors. Erasures are easier to correct than errors at unknown places and make the methods we want to use more effective.

All common FEC codes, including the two we will evaluate, works in the same way when it comes to errors on known positions. You take $k$ symbols of data and use the FEC method of your choice to create $n - k$ symbols of redundant data, giving you $n$ symbols altogether. These symbols are sent to the receiver. The receiver now only has to receive arbitrary $k$ of the original $n$ symbols to be able to recreate all the information.

---

**Figure 3.1 - (4,3) code example of using FEC methods to recover from packet loss**

---

26
Reliable distribution

The symbol size mentioned above is though typically one byte. Therefore some thought has to be put into how to use FEC methods to cover for packet loss, the symbols being of packet size, which is typically a few hundred bytes. What we ideally want is to be able to send \( n \) packets of which \( n - k \) is redundant packets, the packets built in such way that the receiver only have to receive arbitrary \( k \) of these packets to be able to recreate all the information. This is illustrated in Figure 3.1. Sending the FEC data and original information in separate packets is also consistent with the standard proposed in [4].

So how do we accomplish this with the byte-oriented methods available? The method that comes to mind is to still use the methods in a byte-wise way, but to gather the redundant information in packets. If we want to create \( n - k \) redundant packets from \( k \) original packets we can do as follows (Figure 3.2):

- Take the first byte from all the \( k \) packets and perform the FEC encode operation on these bytes.
- Put the \( n - k \) resulting bytes in the first positions of one new packet each.
- Move on to the second byte and do the same thing and continue to the end of the packets.
- When we have gone through all packets, we have \( n - k \) redundant packets giving us \( n \) packets altogether.

![Figure 3.2 - (5, 3) code example of FEC packet encoding](image)

On the receiver side we perform the same procedure except we’re decoding instead of encoding, that is taking one byte at a time from the packets received and decoding these bytes, recreating bytes from lost packets and putting them together to reconstruct the entire packets. To use this method, all packets have to be of the same size. This is where the padding functionality of RTP is usable. We just settle a size that all packets should have, and use padding to make all packets be of this size.

Given the probability, \( p \), of packet loss, it’s easy to calculate the probability that all packets in a set of \( n \) packets can be reconstructed at the receiver. For a given value, \( i \), the factors \((1-p)^i\) and \(p^{n-i}\) gives us the probabilities that \( i \) packets get through and that \( n - i \) packets get lost respectively. The key then is to calculate the probability that at least \( k \) of the \( n \) packets reach their destination:

\[
\sum_{i=k}^{n} \binom{n}{i} (1-p)^i p^{n-i}
\]
**FACTUM**

**Reliable distribution**

With this formula we can, as an example, calculate the probability that all five media packets in a set can be reconstructed using a (8,5) code, given a packet loss probability of 5%:

\[
\sum_{i=0}^{8} \binom{8}{i} 0.95^i \cdot 0.05^{8-i} = 0.997
\]

The method of being able to reconstruct entire packets is well suited to deal with the problem of packet loss since it provides high probability for recovering from the loss of random packets. The latency that this method contributes with is rather modest; we only have to wait for \( n \) packets before we can decode them. The weakness of the method is that it is sensible to burst errors, meaning that several packets in a row get lost. If packets are lost in a sequence greater than the value of \( n - k \) within a set of \( n \) packets, we are unable to repair any of these packets.

### 3.3.2. XOR

The XOR-method, described in [4], is perhaps the simplest FEC method available, simple meaning easy to implement due to simple mathematics and modest hardware performance demands by the same reason. The idea is to for every \( k:th \) packet sent, construct a redundant packet of these \( k \) packets by performing bit-wise XOR-operations on them. The result from this operation forms a redundant packet of the same size as the rest of the packets. The sender sends all of these \( k + 1 \) packets to the receiver. If one of the original packets is lost during transmission, the redundant packet can be used to recreate it, performing bit-wise XOR on the packets that reached the receiver and the redundant packet. This is illustrated in Figure 3.3.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet 1</td>
<td>1011</td>
</tr>
<tr>
<td>Packet 2</td>
<td>0111</td>
</tr>
<tr>
<td>Packet 3</td>
<td>0101</td>
</tr>
<tr>
<td>XOR</td>
<td>FEC packet 1001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet 1</td>
<td>1011</td>
</tr>
<tr>
<td>Packet 3</td>
<td>0101</td>
</tr>
<tr>
<td>FEC packet</td>
<td>1001</td>
</tr>
<tr>
<td>XOR</td>
<td>Packet 2 0111</td>
</tr>
</tbody>
</table>

**Figure 3.3 - The XOR method. Packet 2 is lost during transmission and gets repaired at the receiver.**

The overhead costs for the XOR-method is completely dependant of \( n \), since \( k \) always equals \( n - 1 \). The overhead bandwidth cost is calculated as

\[
\frac{n}{k} = \frac{n}{n-1}
\]

### 3.3.3. Reed Solomon

The Reed Solomon method, described in [6] and [8], is a widely used method for protecting information in all sorts of formats. For example, this is the method used to protect the sound on audio CD’s, so that they are playable even if scratches appear on the surface. It is part of a code family called cyclic codes, who in turn is a subset of codes called block codes. Cyclic codes have the properties of being cyclic as well as linear. Linear means that the sum of two codewords is also a codeword, and cyclic means that the shift of a codeword is also a codeword. The block codes have the property of being independent of previous codewords, as opposed to convolution codes that are calculated taking prior information into consideration. The latter property is important for our purpose, since we want to be able to start receiving
Reliable distribution

information from a sender at any time, and therefore want to be independent of what was sent before we started receiving.

The Reed Solomon algorithm that should be used in our case only has to deal with erasures, that is we know the exact location of the data missing. A general Reed Solomon code can correct both errors and erasures, but the algorithm that can only handle erasures, called RSE (Reed Solomon Erasure correcting code), is a bit less complicated than a general one. The Reed Solomon encoding procedure is briefly described below. All operations are carried out using field arithmetics.

Consider a codeword, C, which is made up of n numbers:

\[ C = (c_{n-1}, c_{n-2}, \ldots, c_0) \]

This can be represented mathematically by a polynomial of degree n.

\[ C(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_0 \]

A code word, C(x), is constructed to be a polynomial of degree n, which can be divided by a generator polynomial g(x) of degree n-k, where

\[ g(x) = (x-a^1)(x-a^2)\ldots(x-a^{n-k}) \]

and a is defined using field arithmetics.

To construct the coefficients in C(x), k of them is set by using the information symbols we want to transmit. The rest n – k of them are set so that C(x) is divisible by g(x). This gives us a system of linear equations with n - k unknowns to solve. The decoding of the code works in a similar way, once again finding the values of up to n – k missing symbols.

Reed Solomon is optimal, which means that no other method can offer a better protection for a certain overhead cost. Furthermore, relative to other cyclic codes, it’s cost effective when it comes to hardware performance demands and was recommended by [7] for the current application.

The overhead bandwidth costs of the Reed Solomon method can be calculated as

\[ \frac{n}{k} \]

3.4. Interleaving

The weakness of FEC methods when exposed to burst errors could be fixed by using interleaving techniques. What interleaving does is to spread information in time so that errors in sequence get spread out in time when the information is interleaved back to its original form. In our case, the deal would be to spread the packets in time, so that if a burst error occurs, causing losses of a big number of packets in a row, we would instead get single packet losses in many places, making it possible for the FEC methods to recreate the lost information.
Reliable distribution

To the left in Figure 3.4, 25 packets that are normally sent from left to right, row by row, are shown. The biggest amount of packets we could lose in a row with this set-up is two packets using the (5, 3) code assumed in the example. If we instead sent the packets in the interleaved order shown to the right in the figure, the sending order still being from left to right, row by row, we would be able to lose ten packets in a row and still be able to correct them with the FEC method.

![Image of packet interleaving example](image)

**Figure 3.4 - (5, 3) example of a group of packets before (left) and after (right) interleaving**

The drawback on interleaving though is pretty serious for our application. For the FEC method to be able to decode the information, it needs access to all \( n \) packets belonging to a set. With the example in the figure, we would have to wait for 21 packets to be transferred before being able to decode anything, this being indicated by the arrow in the figure. For our application this is not suitable at all. We would either have to tolerate a great latency of the audio, or keep the interleaving buffer very small, but then larger burst errors would still be a problem.

### 3.5. Error hiding

No matter what FEC techniques are used to minimize the number of lost packets in a transmission, there will always be a risk of packets somehow getting lost. Therefore, it is necessary to have a strategy for what to do when a packet that is in turn for usage at the receiver is missing. The choices in this situation, assuming that each packet contains a fragment of an audio file, is:

- Skip the packet – will cause a displacement of the audio in time.
- Replace the missing packet with silence or noise of a length corresponding to the length of the sound in the packet.
- Play the contents of the packet preceding the missing one again, hoping they sound about the same.
- Interpolate between previous and following packets, creating a piece of audio well fit to its surroundings.

To use any of the two first alternatives would be stupid, since the third alternative in most cases gives a much better result and furthermore is easily implemented. The interpolation alternative probably gives us the best result, but is far more complicated to implement than the previous ones. It also puts pressure on the hardware, since it needs more calculation resources.

Replacing or interpolating mpeg frames is pretty straightforward, since they are completely independent of preceding and following frames. This is not the case for all mpeg files, but
Reliable distribution
mpeg encoders can be configured to make the frames independent of each other. If one would use such a solution for DAB frames however, some extra calculations would have to be done, since DAB frames contain a checksum calculated for the next frame, and thus aren’t independent of each other. This problem is though easily solved, by simply calculating the checksum of the packet following the missing one.

3.6. Buffering strategies
Since the solution we are about to decide for, using RTP with UDP beneath, doesn’t guarantee that packets reach their destination in the right order, we are forced to provide some kind of buffer at the receiver, letting packets arrive out of order to some extent. Furthermore, if no receiving buffer was used, but the content of the packets was played at the moment they were received, no of the techniques mentioned in this chapter would be applicable. Using a buffer placing every packet in an own slot, we could let there be some space between the position for the next packet to be played and the position for the next packet expected to arrive making it possible for delayed packets with sequence numbers corresponding to positions between these positions to arrive late (Figure 3.5).

![Buffer leaving possibilities for late arrivals](image)

Figure 3.5 - Buffer leaving possibilities for late arrivals

Of course the buffering strategy above gives consequences of an increased delay of the audio, so one would probably try to minimize it considering how late packets are likely to arrive. Note that an advantage we get for free is the possibility to hide errors by replacing missing packets with previous ones, since the previous packets are always available in the buffer.

The buffer at the receiver cannot be infinite, and therefore the sender has to consider at what pace it should be sending its packets. In fact, if the packets arrive any faster than they are played, the buffer would sooner or later overflow. The sender therefore has to know how much time the contents of each packet will take to play, and adjust its pace so that it waits for that period of time between every packet sent. This of course would also be the case if no buffer were to be used.

When using some FEC method, the fact that \( n \) packets have to arrive to the receiver before it can decode anything has to be taken into consideration. This forces us to leave some space between the position of the packet expected to arrive and the position where the decoding algorithm should start (Figure 3.6).
Reliable distribution

![Diagram](image)

Figure 3.6 - Media buffer using FEC decoding, k = 5.

To adjust this buffer to be able to handle packets arriving late, we would have to move both the position for the next decoding and the play position further to the left in accordance with Figure 3.5. The number of positions would have to be a multiple of k, since decoding always has to start with the first packet in a set.

It should be noted that the FEC packets are not shown in Figure 3.6, since these are assumed to be placed in a separate buffer. This buffer would work in the same way as the buffer shown, except there is no play position. The FEC packets, unlike the media packets don’t have to be sent in any special pace by the sender, but should rather be transmitted as fast as possible. The FEC buffer doesn’t risk overflow since packets are arriving and being used in an even pace proportional to the pace of the media buffer.

The smoothest way to implement interleaving is to completely separate this functionality from the functionality discussed above. That way, the functionality behind the buffer for FEC decoding and packets arriving out of order doesn’t even have to know if interleaving is used or not. On the sender side, instead of sending a packet, it would go into an interleaving buffer, and a packet already in the buffer would be sent. On the receiver side, an incoming packet would go into an interleaving buffer, in which the packets are put back in order, and another packet would be pulled out to be handled by the receiver’s main functionality.

### 3.7. The solution of choice

The layer structure of an IP network, which is also discussed in Ch. 2., called the TCP/IP architecture, is shown in Figure 3.7.

![Layer Structure](image)

Figure 3.7 - TCP/IP architecture

Starting with the lowest layer of the protocol stack considered in this work, the network layer, it would be desirable to be able to reserve resources guaranteeing a certain highest delay, minimal packet loss probability and a suitable throughput level. That way it would be easy to configure the solutions of the upper layers to reliably distribute the information. The possibilities to reserve resources (QoS) however, differs between different customers perhaps controlling their networks in varying degrees. Therefore techniques used above this layer shouldn’t make any assumptions, but be configurable to work with different degrees of QoS.

Considering the IP layer, it would be optimal if the network elements were configured to handle IP multicast, since there are great advantages in terms of network resources and
Reliable distribution

accessibility for the receivers won using this. Even though the technology for IP multicast has existed a since long, there is a certain inertia among network owners when it comes to configuring their networks for it. They are sometimes afraid of the consequences it might have on traffic existing on their networks today. Therefore, the solution has to be applicable also to networks not supporting IP multicast.

In the transport layer, UDP is used. To extend UDP with real-time functions, it is complemented with RTP, which could be considered an application protocol. The reliability is extended by adding functions for FEC at the same level as RTP. These functions have to be in the same protocol layer as RTP, which is further discussed in Sec. 3.7.2. The functions for interleaving are put directly beneath the RTP layer, though still being implemented as part of the application.

Above RTP is the application itself, the boundaries between RTP, FEC functionality, interleaving functionality and the application though being somewhat vague. However, the implementation should be made so that it could easily be rebuilt, still keeping the functions for RTP management. The layered structure of the application is shown in Figure 3.8.

<table>
<thead>
<tr>
<th>Application (Sender/Receiver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTP/FEC</td>
</tr>
<tr>
<td>Interleaving</td>
</tr>
<tr>
<td>UDP</td>
</tr>
<tr>
<td>IP (Multicast)</td>
</tr>
<tr>
<td>Unspecified network</td>
</tr>
</tbody>
</table>

Figure 3.8 - Layered structure of the application

3.7.1. Using RTP

Our application will use a subset of the functions provided by RTP. The RTCP protocol used to control the traffic will not be used, since for our application little is gained by getting information about how the transmission is doing. The only fields in the RTP header (Sec. 2.4) that will be used are the following:

Padding
To be able to create packets of the same size no matter the size of the payload.

Payload type
To be able to distinguish the type of payload a packet contains. This field is set mainly to stick with the standard.

Sequence number
Used to detect packet loss and make sure that the incoming packets are handled in the right order.

Timestamp
These values will be set by the sender to stick with the standard, but not used by the receiver, since we already know the differences in time between packets.

SSRC
Will be set to be able to demultiplex media and FEC packets if they should be sent to the same IP address and the same port.
3.7.2. Integrating the FEC methods with RTP

To integrate FEC with RTP a choice has to be made whether to distribute the FEC information and media information in the same packets, or to send the different kinds of information in separate packets. In accordance with our discussion in Sec. 3.3.1, the choice has practically already been made to separate the two. This is the assumed method to use in [12], which further motivates the choice. When using this method, the proposed standard in [4] suggests that media packets and FEC packets are distributed in different streams, the FEC information also being distributed as RTP packets. The streams could be separated by sending the packets to different IP-addresses, different ports or by having different values in the RTP SSRC field. The different streams should also have separated sequence numbering. The separation of the media packets and FEC packets makes it possible for hosts not having techniques for FEC decoding to still make use of the media packets by just receiving the media stream.

Since the FEC encoding/decoding is dependant of the RTP sequence numbers, this functionality cannot be placed beneath RTP in a layered structure. On the other hand, the FEC functions encode/decode complete RTP packets with media contents and thus need to access these packets. Therefore, the FEC functionality cannot be placed strictly above RTP in a layered structure. Altogether, the FEC functionality and the RTP protocol have to be on the same level, as shown in Figure 3.8.

The creation of FEC-packets should be done by the following rules described in [4] and shown below, who are to be considered as minimum demands, since further use could be made of the FEC header.

- Each FEC packet contains a bitmask of 24 bits. If the $i$:th bit in the bitmask is set, $i$ being a number between 1 and 24, this indicates that the media packet with sequence number $N + i$ is one of the packets used to generate the FEC packet, $N$ being a value also supplied by the FEC packet.
- The FEC packets are not sent in the same stream as the media packet.
- The FEC packets have their own sequence numbering.
- The timestamp-value of the FEC packet should be the value of the real-time clock at the sender side in the moment the packet is sent.

A FEC packet is constructed by placing a FEC header and the FEC data in a RTP packet as shown in Figure 3.9.

<table>
<thead>
<tr>
<th>IP Header</th>
<th>UDP Header</th>
<th>RTP Header</th>
<th>FEC Header</th>
<th>FEC Data</th>
</tr>
</thead>
</table>

Figure 3.9 - FEC packet structure

The FEC header consists of 12 bytes and is built as shown in Figure 3.10, with the meaning of its fields described below

<table>
<thead>
<tr>
<th>SN base</th>
<th>Length recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>E PT recovery</td>
<td>Mask</td>
</tr>
<tr>
<td>TS recovery</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.10 - FEC header format
Reliable distribution

SN-base
The lowest sequence number of the media packets that the FEC packet protects.

Length recovery
Used to determine the length of any reconstructed packet.

PT recovery
Used to determine the payload type of any reconstructed packet.

Mask
Contains 24 bits. If bit $i$ is set, this means that the media packet with sequence number $SN_{base} + i$ is protected by the current packet.

TS recovery
Used to determine the timestamp value of any reconstructed packet.

The standard suggests that the FEC method is performed only over the payload of the RTP packet to protect, thus not including the RTP header of the packet. However, some important values of the RTP header have to be repairable, and that is why these values can be encoded in the same way as the payload, and inserted in the recovery fields of the FEC header. The Mask field of the FEC header is not used in our solution, since there is no doubt what FEC packets belong to what media packets; we derive this from the sequence numbers of the RTP packets. The FEC header is very general and really not of much use for us, since for instance the length and payload type of our media packets are constant. However, the FEC header is used to stick with the standard.
4. Prototype

The aim when starting the work of this thesis was to construct a prototype to test the different techniques presented. Since the main problem is to distribute audio over an IP network, the ideal prototype would consist of a server sending the audio and a client receiving the audio and also making it visible to the user by playing it. This becomes possible since there already exists a mpeg player able to receive RTP traffic. This player is called Zinf [20], former known as Freeamp. The source code for this software is free, giving us the possibility to extend it with features making the transmission more reliable. Zinf plays mpeg files including mpeg 1 layer II used by the DAB standard. The format transmitted by the prototype will therefore be mpeg data instead of DAB data, but since DAB basically consists of mpeg data the result will be the same.

Visual C++ was chosen as the implementation environment of the sender, since it is the common choice for Factum projects. Fortunately it turned out that there was source code available to compile the Zinf program in Visual C++, which made it convenient to also implement the intended extensions for Zinf in this environment. The implementation was done in an object oriented way and the reader should therefore be prepared for that object oriented concepts such as classes and objects may occur in this chapter.

In this chapter the server (sender) and the client (receiver) will be described. There will not be a complete description of the implementation, but some important implementation issues will be discussed.

4.1. Sender

User interface
The user interface of the sender contains a number of configuration possibilities for how a file should be transmitted (Figure 4.1). First, one has to choose a file to send and an address that tells the program where to send the file. In the area with the title “FEC” one can configure what FEC method should be used and the properties for this method. One also has to choose whether to use interleaving or not. In the area marked “Simulation”, configurations can be made to simulate different kinds of packet loss at an optional packet loss rate. The linear method linearly loses every i:th packet, i depending of the percentage given. The random method randomly loses packets though still keeping the rate at the level given by the percentage. If the burst error method is selected, errors will appear in bursts as long as the percentage given, that is a packet loss rate of 15% will occasionally lose 15 packets in a row.
Prototype

To start the transmission, the Send-button is pressed, making the same button read Stop, which is pressed to stop the transmission.

![Prototype Interface](image)

**Figure 4.1 - User interface of the sender**

**Reading the file**

When the Send-button is pressed, the chosen file will be opened. Mpeg files are built so that a number of synchronization-bits indicate where the first mpeg-frame starts, in fact all frames start with these synchronization-bits. However, the program starts scanning the beginning of the file looking for these synchronization-bits, to avoid starting to process unwanted information that could be stored there. When the synchronization-bits are found, the mpeg header is read and interpreted so that we know the size of the mpeg frame we are about to read. We are now ready to provide the rest of the program with mpeg frames. The functionality of the class that handles file reading is abstract to the other classes in the program, just knowing they are able to request mpeg frames from it. This makes it possible to easily exchange this class for something else providing the program with mpeg frames, for example a real-time mpeg encoder.

**Constructing FEC packets**

The main routine running in the program requests mpeg frames from the class described above. When the RTP packets payload, the size of the payload being specified by a constant, is filled with an even number of mpeg frames, a RTP packet is constructed and sent, but also stored. When $k$ RTP packets are stored, these are sent to a routine that creates FEC packets. The FEC routine takes a byte from each packet and performs a FEC operation on these bytes. If the method chosen is XOR, the operation to perform is simply bit-wise XOR. If the method is Reed Solomon, the bytes are sent to the functions handling the Reed Solomon calculations.
Prototype

The Reed Solomon code wasn’t implemented in this project, but found on the Internet [15] and converted from C to C++ format. When the FEC packets have been constructed in accordance with the description in Sec. 3.3.1 they are sent immediately after each other.

Timing

The sending of packets has to be timed so that the buffers at the receiver don’t get overflowed. The packets should therefore be sent with a delay corresponding to the lasting time of their audio contents. This time is interpreted from the mpeg headers and of course depending of how many mpeg frames that fit into one RTP packet. But the operations that are performed on the sender mpeg frames are somewhat time consuming. Therefore, these operations are timed, and after having sent a packet the program waits for a time equal to the difference between the lasting time of the packet and the time that the operations has consumed before sending another packet.

Interleaving

Interleaving should be the last thing to be done before sending the packets. The rest of the functionality in the program doesn’t even have to know about interleaving being done. Therefore, when interleaving is activated, the packet is sent to the interleaving buffer instead of being sent to the receiver, and a packet from the interleaving buffer is sent to the receiver. In this prototype the interleaving depth is always set to $n$ for simplicity. The storing of the packets is done in a matrix of the size $2 \cdot n^2$, which makes it possible to read in one half of the buffer and write in the other half. Media packets and FEC packets are stored in the same buffer. The packets are stored after each other row-wise in the matrix, and when they are sent, this is done in a column-wise order (Figure 3.4). This way, the packets get sent in the order $1, n+1, \ldots, n, 2n, \ldots, n^2$.

4.2. Receiver

User interface

The interface of the receiver is mainly the original interface of the Zinf software. Some statistical numbers are added in the main window. As illustrated in Figure 4.2, the third row in the status window shows how many packets have been fixed by the error correcting methods, how many packets that should have arrived in total, and how many packets the program haven’t been able to correct due to packet losses greater than $n - k$ in a set of $n$ packets.

![Figure 4.2 - User interface of the receiver](image)

The receiver, like the sender, has to be configured to know which FEC method to use, the specifics of this method and whether to use interleaving or not. This is done by pressing the button named “Files” in the main window (Figure 4.2). This launches a new window (Figure...
Prototype

4.3) where configurations can be made. The input in the dialog in the field named “URL” should follow the pattern


When this is filled out, the button named “Open URL” is pressed and the receiver starts listening for incoming RTP packets. The <replace> parameter tells the program whether to use error hiding or not, the method used being to replace unrepairable packets with packets received earlier. If error hiding is not used, unrepairable packets are simply skipped.

![Add Tracks and Playlists](image)

Figure 4.3 - Dialog for configuration of the receiver

Buffering

The buffering strategy for the incoming packets has to consider the facts discussed in Sec. 3.6. Since we have the advantage of the packets being numbered when they arrive, we are able to use modulo operations to put them in the right place in the buffer. This way the arriving order of the packets doesn’t matter as long as they lie within the boundaries of the buffer. It also makes it possible to use a bounded buffer, which can be looked on as a circular buffer filling the circle with data from one position and starting over at that position when the buffer is full, using modulo operations. The operation for finding out in which position to put a packet is calculated by the following formula:

\[
\text{position} = \text{sequenceNumber} \mod \text{BufferSize}
\]

The FEC packets are stored in their own bounded buffer partly because they are of different size, but also for the program to be able to forward packets to the original Zinf software from the media buffer without having to “jump” over FEC packets. In the current implementation, the decoding position is always between \(k\) and \(2k\) positions behind the position of the expected incoming packet, leaving a space of at least \(k\) positions for packets arriving out of order. The play position is at a minimum distance, varying between \(0\) and \(k\), behind the decoding position (Figure 4.4). These positions are easily modified if one for example would like to extend the possibilities for packets arriving out of order. Packets with sequence
numbers giving them positions before the decoding position are too old to be considered and are discarded.

Figure 4.4 - Media buffer. Delayed packets fitting in the interval 17 - 24 are able to arrive. \( k = 5 \).

The decoding algorithm is started when all FEC packets of a set have arrived, or when more than \( k \) media packets have arrived since the decoding algorithm was run the last time. The playing of packets is triggered when a new packet arrives. Every time a packet arrives, a packet is taken from the buffer and played. To cope with packet loss though, something has to be done to keep the buffer in synch. When a packet arrives, we calculate how many packets are lost since the last packet we received using the sequence numbers, and thus play number of missing packets + 1 every time a packet arrives.

Timing
In the receiver, the mpeg frames have to be played at a certain rate independent of variability of delay in the network. This functionality was already implemented in the Zinf software, which provide a buffer from which frames are played at an even pace.

FEC decoding
The decoding of the packets is done in a similar way to the encoding procedure described in Sec. 4.1. One byte at a time is taken from the packets in a set and put together forming \( n \) bytes, bytes missing due to packet loss replaced with the value of zero. These bytes are decoded either using simple bit-wise XOR operations or by using the Reed Solomon code. When the entire packets have been gone through, we have rebuilt the missing ones. It should be noted that the FEC decoding procedure is not performed if no media packets are missing or if more than \( n - k \) packets are missing in a set, making it impossible to repair the lost packets.

Interleaving
At the receiver side, interleaving is not as straightforward as on the sender side. This is mainly because we have to deal with packet loss and thus have to consider the sequence numbers of the RTP packets. The fact that media and FEC packets are distributed in different streams makes this a bit complicated. As mentioned earlier, the application is implemented so that the interleaving depth is always \( n \). This implicates that all \( kn \) media packets will arrive before the \( (n-k)n \) FEC packets arrive (Figure 4.5).

Figure 4.5 – (5,3) example of interleaved packets. The FEC packets are colored grey.
Prototype

Since the media packets and the FEC packets have different sequence numbering, the packets are placed in different buffers, which makes the work of sorting the packets before they are forwarded to the major routines in the receiver a bit more difficult.

The storing buffers consist of one buffer for the media packets and one for the FEC packets with the size of \(2 \cdot k \cdot n\) and \(2 \cdot (n-k) \cdot n\) respectively. The double size of the buffers makes it possible to read from one half of the buffers and write in the other half. To be able to decode a set of interleaved packets we have to receive a major part of it. But we want to be able to start receiving packets whenever we want to, without having to synchronize this with the sender. This means that we have to wait with the decoding until we get a good start of a set of packets; trying to decode a set of packets with only FEC packets is not a good idea for instance (Figure 4.6). The functionality for this is implemented in a heuristic manner: the interleaving class doesn’t start forwarding packets until it has received a set of packets where at least half of the media packets are present. When such a set has been received, the forwarding of packets starts.

![Figure 4.6](image)

Figure 4.6 – (5,3) example of reception, started at an arbitrary moment with occurrences of packet loss.

The sorting of the packets from the incoming interleaved order to their original order is done in a way similar to the method described in the buffering section. The packets are inserted in the buffers at places calculated by performing modulo operations on their sequence numbers.

**Error hiding**

The error hiding feature, implemented as packet replacement, takes place immediately before the packet is supposed to be played. If the play position points at a position that doesn’t contain a valid packet, the packet from the previous position is copied into this position and played. The copying ensures that a previous packet to the current packet always exists.
5. Testing

When the prototype was finished it had to be tested. First of all, of course a test had to be made to see that the multicasting distribution of music worked. This was tested by running the sender and the receiver on different computers in the same network. This part of the application turned out to work fine. The rest of the tests were mainly performed by running both the sender and the receiver on the same computer, making it multicast IP packets to itself.

5.1. Packet loss recovery

Some testing has been done to measure the performance of the methods using some various configurations of the n- and k-values. The packet loss method used in these tests were the random method, which is the most realistic one. The packet loss-simulation is implemented in the sender as explained in Sec. 4.1. If the random function decides to drop a packet, it simulates this by simply not sending the packet. The results of the tests are shown in Table 5.1. The “Loss” column in the table is the percentage of packets that are dropped in the simulation; these do not reach the receiver. The columns with the title “Share of unrepaird packets” show the share of packets that the FEC method in the receiver hasn’t been able to correct. Three tries were done for each method and a mean value was calculated. The “Improvement” column shows the improvement of playable packets in percentage points compared to the probability of packets arriving based on the loss-value:

\[ \text{ Improvement } = (1 - \text{ Mean}) - (1 - \text{ Loss}) \]

<table>
<thead>
<tr>
<th>Method</th>
<th>Loss (%)</th>
<th>Share of unrepaird packets</th>
<th>Improvement (%)</th>
<th>Delay (ms)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Try 1</td>
<td>Try 2</td>
<td>Try 3</td>
<td>Mean</td>
</tr>
<tr>
<td>XOR (5,4)</td>
<td>5</td>
<td>0,009762</td>
<td>0,006746</td>
<td>0,010238</td>
<td>0,008915</td>
</tr>
<tr>
<td>XOR (3,2)</td>
<td>5</td>
<td>0,003</td>
<td>0,004667</td>
<td>0,008333</td>
<td>0,005333</td>
</tr>
<tr>
<td>RS (8,5)</td>
<td>5</td>
<td>0,000476</td>
<td>0,000159</td>
<td>0</td>
<td>0,000212</td>
</tr>
<tr>
<td>RS (10,6)</td>
<td>5</td>
<td>0</td>
<td>0,000167</td>
<td>0,000167</td>
<td>0,000111</td>
</tr>
<tr>
<td>XOR (5,4)</td>
<td>10</td>
<td>0,031</td>
<td>0,035333</td>
<td>0,034</td>
<td>0,033444</td>
</tr>
<tr>
<td>XOR (3,2)</td>
<td>10</td>
<td>0,019286</td>
<td>0,021333</td>
<td>0,019667</td>
<td>0,020095</td>
</tr>
<tr>
<td>RS (8,5)</td>
<td>10</td>
<td>0,002</td>
<td>0,0015</td>
<td>0,004667</td>
<td>0,002722</td>
</tr>
<tr>
<td>RS (10,6)</td>
<td>10</td>
<td>0,002</td>
<td>0,0015</td>
<td>0,001143</td>
<td>0,001548</td>
</tr>
</tbody>
</table>

Table 5.1 - Statistics of improvement using FEC
Testing

No statistics are presented of how burst errors affect the correction of packets, since this is predictable. If we get a burst error of ten packets using a (8,5) code for instance, between seven and ten of these packets will not be repairable, depending of how the missing packets are situated. Error hiding doesn’t cover for errors of this size, and the music is not enjoyable if burst errors occur frequently. When applying interleaving however, the application gets far more tolerable of burst errors, but also this result is predictable as described in Sec. 3.4.

The statistic result presented above might be interesting to look at when choosing which FEC method to use. However, a subjective test of the sound quality of different methods should be appropriate, since this is what really matters in a real application. Such a test was performed with the results given in Figure 5.1. The rating was done by one person listening to three different kinds of songs for each method, the grade given being the summarized impression of the sound quality of these three songs played by the receiver. Error hiding was used during the test to improve the sound quality.

![Sound quality chart](image)

**Figure 5.1 - Subjective measurement of sound quality**

The rating in Figure 5.1 ranging from 1 to 5 has the following criteria:

1 – Unable to recognize the song.
2 – The song is recognizable, but there is too much interference to enjoy it.
3 – The song is enjoyable but the disturbances are noticeable.
4 – Interferences occur, but are only recognized during concentrated listening.
5 – Perfect sound quality.

### 5.2. Hardware load

A significant issue from the users point of view is the amount of hardware load the applications cause. This is important since the multiplexers might have to receive and thus decode signals from several senders simultaneously. Therefore it is interesting to see how much load decoding one signal causes a processor with similar performance as the ones used in a multiplexer. The sender software has a modest demand for hardware resources and was
## Testing

Therefore not tested. The test was done by measuring the mean value of the percentage of the processors resources that the receiver process demands, using different kinds of FEC methods and packet loss rates on a computer with a Pentium II 400MHz processor.

<table>
<thead>
<tr>
<th>Method</th>
<th>5 % packet loss</th>
<th>10 % packet loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR (5,4)</td>
<td>0.2 %</td>
<td>0.25 %</td>
</tr>
<tr>
<td>XOR (3, 2)</td>
<td>0.25 %</td>
<td>0.29 %</td>
</tr>
<tr>
<td>RS (8,5)</td>
<td>24 %</td>
<td>39 %</td>
</tr>
<tr>
<td>RS (10,6)</td>
<td>26 %</td>
<td>49 %</td>
</tr>
</tbody>
</table>

Table 5.2 - Results of hardware load test

The results presented in Table 5.2 make it obvious that the demands on hardware performance using XOR methods are neglectable, while using Reed Solomon needs significant hardware resources and would require a top of the line machine if it should be able to decode several streams of IP traffic. This is also dependent of what rates of packet loss one can expect from the network. The load is highly dependent of this, since the more often packets are lost, the more frequently the software has to recreate packets using calculations. The great differences between the two methods can to some extent be explained by the fact that the Reed Solomon method has to solve equation system with several unknown values, while the XOR can only handle one unknown value, but also because there is more advanced mathematics within the calculations performed by the Reed Solomon method.

### 5.3. IP over DAB experiment

As mentioned in Sec. 1.3 there is some interest in combining the work in this project with IP over DAB technology. To test if the FEC methods developed for DAB over IP would provide any help when streaming audio using IP over DAB, a test environment was set up using the prototype applications developed in this project. The sender distributed its packets to a multicast group. A converter received these by joining this group and converted the IP traffic into a DAB signal and sent this to a DAB transmitter, who transmitted the signal into the air. The receiver was installed at a computer who received IP traffic converted from a DAB receiver that was connected to it (Figure 5.2).

![Figure 5.2 - IP over DAB experiment](image)

When the environment was set up with the DAB receiver fairly close to the DAB sender the playback of the song was carried out without interference. The distance between the DAB receiver and the DAB sender was then increased, so that the signal became weaker. This was done to see if any use could be made of the FEC methods for packet loss. It became clear pretty soon though, that this was not the case. As soon as the receiver didn’t get a clear signal from the sender, there was significant packet loss of such amount that the FEC software could not repair it. This probably has to do with the frequent appearance of bit errors in a weak signal, since the UDP protocol silently drops packets containing these kind of errors. It can therefore be stated that the methods developed in this project are not suitable to improve the IP over DAB technology.
6. Results

The results drawn from this project are of significant value when deciding which methods to use when distributing digital radio over an IP backbone. It becomes clear that this is fully possible, provided the underlying network can be configured to have reasonable amounts of packet loss with burst errors being a phenomena that never or very seldom occur. All other disadvantages within an IP network has shown to be manageable by software. The possibility of using the methods on a unicast network hasn’t been implemented in the application, but shouldn’t demand many modifications. The sender would just have to send its packets to multiple addresses instead of one, and the receiver would listen to an ordinary IP address instead of joining a multicast group. Interleaving is simply not recommended due to its disadvantage of increased delay (Sec. 6.1). Some demands of what should be asked for from the underlying network are summarized below:

- The network should preferably be configured for multicast traffic.
- Packet loss rate should be as low as possible and not higher than 10%.
- Burst errors should be avoided or occur very seldom, since this unconditionally leads to interferences in the sound.

The major software problem seems to be the hardware load when using advanced codes as Reed Solomon. This problem could to some unknown extent be reduced by optimizations, but will probably remain the main disadvantage. The XOR method isn’t as effective as Reed Solomon, but should be usable at a low rate of packet loss, that is a rate under 5 %, which isn’t too much to ask for from an arbitrary IP network. Since the XOR method has very modest requirements considering hardware performance, this probably would be the best method to implement among the ones tested.

Problems with overhead are of course highly dependent of the bandwidth performance of the network where the application should be implemented. As illustrated in Table 5.1, the amounts of overhead have to be significant if a good result should be obtained. The cost of increased bandwidth demands due to overhead though seems to be worth the price considering the corresponding decreased effects of packet loss.

6.1. Delay

The delay of the signal is a major issue of the result, since DAB is very sensitive to this. The tests show that it is possible to keep the delay within the range of a few hundred milliseconds.
**Results**

**Buffering**
The buffering strategies discussed in Sec. 4.2 gives a delay of $k$ times the duration of the audio in a packet. This delay could be avoided if packets arriving out of order were ignored and instead considered lost. However, the buffering necessary to perform FEC decoding needs to be kept, but contributes with rather small values of delay. Using a (3,2) code for instance and the duration of the audio being 24 ms per packet, 24 ms being the common value for mpeg I layer II frames, the delay caused by FEC decoding would be less than 100ms.

**Interleaving**
Using interleaving has the big disadvantage of causing a delay of the signal not suitable for a real-time application. Therefore it would probably not be implemented in a real application. This means that a solution will still be vulnerable to burst errors, and this will in some way have to be handled by functionality in lower layers of the protocol stack.

**6.2. QoS**
It has been established that for our solution to work in the way we want, we will either have to use a network dedicated only for our traffic, or some kind of Quality of Service has to be provided by the underlying network. We namely have to know some facts about the network to be able to configure our software properly. Using DiffServ or IntServ would be an appropriate solution to this. However, issues of QoS have not been tested, since no network with the characteristics of a network suitable for our solution has been available for testing.

**6.3. Possible optimizations**
If the software constructed in this project should ever be used in reality, that is to actually distribute digital radio over an IP backbone, it would probably have to go through some optimizations to ensure that the available hardware can run it without any problems. This part gives some pointers to the routines where optimization could be done.

**Padding**
The padding of RTP packets clearly is an issue for optimization, since the padded bytes in the end of a packet is treated as if they contained real information by the error correcting routines. This of course puts unnecessary load on the hardware. The size of RTP packets is constant and set in the sender and the receiver. One could either adjust this value to fit an even number of mpeg frames exactly in the payload, making sure no frames in the mpeg file differs from the presumed value, or use some kind of faked padding not even having to actually transmit padded packets, but add a minimum number of padding bytes just before encoding and decoding. This would also save bandwidth resources.

**Memory management**
The code has been constructed without any thought of how the hardware is constructed. To some extent this is handled by the compiler, but great amounts of processor work could probably be saved by rewriting the code on sensitive places to suit the hardware better, for example by making sure the memory in the cache is used in the highest degree possible.

**Reed Solomon algorithm**
The algorithm used for Reed Solomon decoding has not been optimized for our purposes. The code is written to be able to both correct errors and erasures. In [8], it is said that an algorithm dedicated for correcting erasures only, which is all we really need, is a lot easier to implement.
Results

and would thus cause less hardware load. This is probably the most important aspect to study if an application similar to the prototype developed for this project should be implemented in reality.
7. References

[1] Goncalves & Niles
   IP Multicasting – Concepts and Applications
   McGraw Hill 1999

[2] Pete Loshin
   TCP/IP Clearly Explained
   Morgan Kaufmann 1999

[3] IETF
   RFC 1889 - RTP: A Transport Protocol for Real-Time Applications
   http://www.ietf.org/rfc/rfc1889.txt

[4] IETF
   RFC 2733 - An RTP Payload Format for Generic Forward Error
   http://www.ietf.org/rfc/rfc2733.txt

[5] IETF
   RFC 2354 - Options for Repair of Streaming Media
   http://www.ietf.org/rfc/rfc2354.txt

   Error Control Systems
   Prentice Hall 1995

[7] Tomas Ericsson, ISY

[8] Anthony J. McAuley
   Reliable broadband communication using a burst erasure correction code
   http://citeseer.nj.nec.com/mcauley90reliable.html

[9] Gunnar Blom
   Sannolikhetssteori och statistikteori med tillämpningar
   Studentlitteratur 1989

[10] Factum Electronics AB
    Factum DAB System Description
    Factum Electronics AB 2002
References

*Computer networks – a system approach*  
*Morgan Kaufmann* 2000

[12] IETF  
*An RTP payload format for Reed Solomon codes*  

[13] IETF  
*RFC 768 - User datagram protocol*  
http://www.ietf.org/rfc/rfc768.txt

[14] Mpgedit.org  
*MPEG frame format*  
http://mpgedit.org/mpgedit/mpeg_format/MP3Format.html

[15] Sourceforge.net  
*Reed Solomon Code*  
http://sourceforge.net/projects/rscode/

[16] Van Jacobson  
*Moving the Internet beyond best-effort*  

[17] Lixia Zhang  
*Resource management and differentiated services*  
http://www.internet2.edu/qos/may98Workshop/presentations/Zhang/

[18] IETF  
*RFC 1112 – Host extensions for IP multicast*  
http://www.ietf.org/rfc/rfc1112.txt

[19] Jim Gogan  
*IP Quality of Service: IntServ and DiffServ*  
http://www.unc.edu/~gogan/qos/sld001.htm

[20] The Zinf project  
*Zinf audio player*  
http://www.zinf.org/
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell förskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances. The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Henrik Brunberg]