Final thesis

Tradeoffs between retransmission and forward error correction in the RTP stack

by

Erman Döser

LITH-IDA-EX-2014/A-14/056-SE

2014-10-22
Final thesis

Tradeoffs between retransmission and forward error correction in the RTP stack

by

Erman Döser

LITH-IDA-EX-2014/A-14/056-SE

2014-10-22

Supervisor: Niklas Carlsson and Morgan Lindqvist

Examiner: Jose M. Peña
Abstract

Video conferencing applications has reached worldwide usage in recent years by the help of the improvements in network infrastructures for public services. Media data covers a significant ratio of data traffic over IP networks. However, it is challenging to ensure a decent quality of service (QoS) on public networks in terms of video and audio quality.

The main factor that may cause degradation in media playback quality is packet losses. There are various techniques available to conceal packet losses in lossy channels. According to the application needs and channel characteristics such as loss patterns and round trip times, retransmission or forward error correction techniques may be applied at application level. These two techniques have different challenges which lead to tradeoffs between them, thus one might be chosen over the others.

In this thesis work, retransmission’s worst case performances under considered packet loss patterns and various round trip times are compared to performances of forward error correction schemes. In addition, implementation details with respect to the relevant RFCs are provided as an example to give a better judgement on the obtained results.

Results obtained under the packet loss patterns that are generated with a simple Gilbert-Elliot 2-state model shows that forward error correction techniques are a reasonable choice of error concealing in the real-time transport protocol (RTP) stack where round trip time in the channel is greater than 200 ms. In addition, bandwidth overhead revealed by forward error correction stays higher than retransmission’s bandwidth overhead in all sample runs. In cases where round trip times are high, then the choice of forward error correction scheme is bound to the packet loss pattern. In the results section, it is obtained that ReedSolomon performs well in terms of residual packet losses, which are the packets not being recovered, and bandwidth overhead when losses occur in long bursts.
Acknowledgements

I would like to express my appreciation to my supervisor Morgan Lindqvist at Ericsson Research for helping me to reach deeper understanding in the subject with useful discussions and his valuable guidance, and to Niklas Carlsson and Jose M. Peña for keeping me in academic direction and their insightful comments. Moreover, I am very thankful to everybody at Ericsson Research for lecturing me and discussing with me whenever I had run into technical obstacles. In addition, I would like to thank my lovely mom, sister and all my close friends for their support and belief in me during my thesis work and life in general.
# Contents

1 Introduction  
  1.1 Problem Statement  
  1.2 Forward Error Correction  
  1.3 Retransmission  
  1.4 Contributions  
  1.5 Limitations  
  1.6 Thesis Structure  

2 Background  
  2.1 RTP  
  2.2 RTCP  
  2.3 Multiplexing Schemes  
    2.3.1 SSRC Multiplexing  
    2.3.2 Session Multiplexing  
  2.4 Forward Error Correction in RTP  
    2.4.1 Parity FEC  
    2.4.2 Reed-Solomon Codes  
      2.4.2.1 Galois Field Arithmetic  
      2.4.2.2 Vandermonde Matrices  
      2.4.2.3 Gauss-Jordan Elimination for Vandermonde Matrix Inversion  
  2.5 System Description  
    2.5.1 Test Application  

3 Methodology  
  3.1 Implementation  
    3.1.1 Parity FEC Encoder and Decoder  
    3.1.2 Reed-Solomon Encoder and Decoder  
    3.1.3 Fec Config API  
  3.2 Network Evaluation  
    3.2.1 Generating Packet Loss Models  
    3.2.2 Use of NetDisturb  

4 Results

vii
Contents

5 Discussion 36
6 Future Work 38
List of Figures

1.1 RTP over UDP communication. ........................................... 2
1.2 One-way transmission of FEC packets. t(N): timestamp. .... 4

2.1 RTP Header .............................................................. 7
2.2 Generic FEC Header ?? .............................................. 11
2.3 Types of Parity FEC Schemes. ....................................... 13
2.4 ReedSolomon FEC Packet Header ?? ............................. 17
2.5 Wireshark capture of the packets generated by the test ap-

lication. ...................................................................... 18

3.1 RTP Stack with FEC modules. ........................................ 20
3.2 Class diagram for FEC encoders and decoders. .............. 21
3.3 Source block generation for ReedSolomon encoder. ........ 23
3.4 Burst loss distribution for the trace between Amherst, Mass-

achusetts - SICS, Stockholm with loss rate 1.9%. ?? ....... 24
3.5 Simple Gilbert-Elliot 2-state Markov model for error process.

[?] ................................................................. 26
3.6 Burst distribution of the trace generated by implemented
Gilbert method. p = 0.01923 and r = 0.80123 where max_burst_length
= 4. ....................................................................... 27
3.7 NetDisturb configuration overview for packet loss simulation. 28
3.8 NetDisturb client window during the simulation showing the
throughput and the packet loss at the sampled time. .......... 29

4.1 Retransmission worst-case performance. ...................... 32
4.2 Retransmission worst case performance when RTT = [99.1-

200] ms vs. FEC algorithms. ............................................. 33
4.3 Detailed look of Figure 4.2 ............................................. 33
4.4 Retransmission worst case performance when RTT = [65.3-

99.0] ms vs. FEC algorithms. ............................................. 34
4.5 Detailed look of Figure 4.4 ............................................. 34
List of Figures

x
Chapter 1

Introduction

The ways of communication keep evolving and being shifted towards the systems that provide more interactivity and better quality such as video conference applications, while the bandwidth of Internet Protocol (IP) networks increases. These applications are being used widely in today's IP networks for personal usage or corporate purposes. For instance, many company meetings are held remotely as video conferences and individuals contact each other through Voice over IP (VoIP)/video chat systems. However, while real-time communication is integrated into our daily lives and increases its importance, coping with network problems such as to deliver reliable data without delays and high quality becomes mainstream.

IP networks are designed to provide best-effort services, which provides best performance available according to the traffic load but does not guarantee packet delivery. Therefore, there is a necessity to recover packet losses in lossy network channels in order to provide high-quality and low latency media streaming. Decision on which method to use in some situations to recover packet losses is depended on network characteristics, hence there is not a perfect loss recovery scheme.

1.1 Problem Statement

In real-time media video/audio data transmission over packet networks, reliable delivery, timing synchronization, and awareness of reception quality of video/audio packets are essential services to achieve synced and high-quality playback at the application layer. However, the Internet, as its nature, does not provide such mechanisms.

Real-time Transport Protocol (RTP) and Real-time Transport Control Protocol (RTCP) are developed to provide application level framing, thus it leaves the decision of network problems to the application control. Furthermore, an application has the capability of implementing the missing mechanisms that are required for a reliable real-time packet transport at a
Chapter 1. Introduction

Figure 1.1: RTP over UDP communication.

certain quality of service (QoS) level. Since RTP is generally placed on top of User Datagram Protocol (UDP) in Open Systems Interconnection (OSI) model to be able to benefit from its checksum and multiplexing features, mechanisms such as participant identification, continuous sequence number space, timing synchronisation and reception quality feedback, as features of RTP and RTCP, carries significance to compensate UDP’s unreliable packet delivery, and arbitrary delivery order over IP networks.

Transmitting audio/video over IP networks is a convenient approach in terms of costs and link utilization since one infrastructure can be used for many purposes and transmission rates are adaptive according to the traffic load. However, packets tend to get lost because of network congestion, signal drops, or faulty network equipments on IP networks. These losses can cause minor or major defects in audio/video quality depending on the loss and media decoder characteristics. This fact leads us to develop techniques to fully recover or minimize the negative effects of packet losses, such as error concealment and error correction techniques.

The focus of this work is on error correction techniques rather than error concealment algorithms, which is mimicking the packet in terms of the lost signal data, and evaluate the performance of retransmission and the chosen forward error correction algorithms in RTP. In addition, the aim is to obtain the tie between the chosen algorithms and various network conditions that provide the highest possible media quality as well as an acceptable number of lost packets that are not recovered during a valid session.

In general, retrieving lost packets by retransmission from the source results performance leaks due to the long round-trip times, in a real time environment. However, retransmission is an excellent method where the round-trip time on the channel is short. On the contrary, forward error correction techniques does not require retransmission of the lost packets, instead it is intended to reconstruct the lost packets on the receiver side by using the redundant bit stream that is already generated and sent by the sender. Thus, the time consumed for traversing the network path to ask for a missing packet and waiting for sender to retransmit the requested
packet is omitted, which in some scenarios prevents retransmission method from satisfying the application’s needs. However, compared to retransmission, forward error correction techniques use more bandwidth due to the redundant packet stream, even though there is no need for correction. The primary challenge in this trade-off is determining the threshold, at where one technique becomes more efficient than the other, and switching dynamically at the obtained threshold for a optimum performance.

This thesis offers thresholds for switching between error correction techniques, as well as different policies of dynamic switching with their benefits and drawbacks in packet networks, especially for real time data transport. There are different error correction codes that have different characteristics such as Parity, Reed-Solomon, Raptor, and Raptor Q, etc. Among various techniques for redundant packet generation, Parity FEC with 1D/2D policies and Reed-Solomon are selected for evaluation in terms of their encoding/decoding speed and complexity, recovery ratios under various packet loss patterns. In addition, the performance of retransmission is evaluated in order to gain deeper understanding for the effect of round trip times on the recovery of packet losses.

Measuring performances of different error correction techniques for real-time media data on a massive IP network like the Internet, has many parameters: Average packet loss rates, packet loss pattern, and traffic load of the network, round-trip times, which is the time spent for a packet to traverse the number of hops between two end points plus the time spent for an acknowledgement packet to traverse the same path backwards, and packet sizes for payloads etc. The results that are obtained from this thesis work are viable to the real-time media applications that use RTP to transmit NAL units of an H264 encoder and subject to change according to the application characteristics.

1.2 Forward Error Correction

Forward error correction is a generic method for recovering lost data by generating the lost bit streams out of successfully received bits and redundant bits that also sent by the receiver. This technique is viable to erasure channels where packet losses are natural. The state of art offers two types of FEC codes: convolutional and block codes. Convolutional codes are applied on a bit streams of various lengths. On the contrary, block codes are applied on bits or packets with a certain size.

The amount of additional bits can be added to a stream is restricted by a maximum transfer rate (MTR) of the medium that data is carried on, due to the bandwidth restrictions. Hence, one can increase the packet loss by adding large amount of FEC because exceeding the bandwidth limits may cause a traffic congestion.

The essential benefit of using FEC to recover packet losses is the scalability because the amount of FEC added to a media stream does not depended
Chapter 1. Introduction

Figure 1.2: One-way transmission of FEC packets. t(N): timestamp.

on the number of users. However, it is depended on the loss rate and the pattern of packet loss. On the other hand, the drawback of FEC in real-time data transmission is the delay that it introduces, since a receiver needs to wait until the necessary packets and the redundancy packet arrives in order to recover the lost packet. Also, a chosen FEC scheme has its own limitation on the amount of lost data it can be used to recover, which can dynamically change during a peer-to-peer session according to some kind of a feedback information on the current loss rate, or as an average of loss rate feedbacks in a multi-cast session [?].

1.3 Retransmission

Computer networks are usually called unreliable networks. Thus, in the cases that a reliable transport of data is required, it should be ensured by the protocol that is used, such as Transmission Control Protocol (TCP). TCP is placed in the transport layer of OSI, and aims for reliable data transportation, not for timeliness. Hence, the lost packets are detected by the protocol itself and resent by using Retransmission with positive acknowledgement. On the other hand, retransmission technique is also suitable for RTP over UDP, where the UDP itself does not guarantee packet delivery but RTP’s features provides feasibility to detect packet losses.

Another way of recovering lost packets is resending them from the sender to receiver upon a feedback on a particular packet’s loss. Retransmission is a very simple way of ensuring packet reliability. The basic concept works as buffering a packet sent from a source for a certain time that is set during the configuration. If a negative acknowledgement for a particular packet is received, the buffered copy is resent to the receiver from the sender. This routine may happen multiple times in case of retransmitted packets are also lost, and as long as sender has a copy of the previously sent message.

Although retransmission works as an on-demand approach, and does not consume unnecessary bandwidth by not sending redundant packets, it may not meet the certain properties of some real-time applications like video conference applications under some network conditions. Especially when the round-trip times exceed a certain threshold between two end points, resent packets from the source point would not be valid any more and shall
be dropped. In addition, in large multi-cast sessions, feedback messages may contribute to the generation of packet losses, by causing a network congestion.

1.4 Contributions

The purpose of this thesis is to obtain the major tradeoffs between two different approaches; retransmission and forward error correction. Thus, two important contributions in the subject of RTP packet recovery are stated. First contribution is drawing a line between retransmission and forward error correction at where one becomes a better choice over another in terms of the likeliness of a possible packet loss recovery. Second contribution is a detailed explanation of packet level forward error correction in the RTP by revealing its difficulties and tradeoffs.

1.5 Limitations

Due to the amount of work done in this thesis work, the algorithm implementations are chosen to be straightforward and may not result in the most efficient computational complexity. This may cause latency for a recovered packet to arrive to the application more than 200 ms if the source block is bigger than the ones used in the test application.

1.6 Thesis Structure

Chapter 1 gives an overview on the problem statement and rapid inside on what approaches could be suitable to address this particular problem. Chapter 2 describes detailed background on the RTP protocol and concepts related to the context of this thesis work. In Chapter 3, the module design, performance tweaks, system overview, chosen loss patterns, and implementation details are stated. As a conclusion, performance measurements of the implemented forward error correction algorithms in contrast to retransmission’s worst cast performance are shown.
Chapter 2

Background

RTP is designed for transmission of data with real-time characteristics. RFC 3550, published by internet engineering task force (IETF), specifies RTP in detail. RTP is not only designed for large-scale video conferencing applications but also for storage of continuous data and distributed interactive applications such as real-time vehicle simulations in virtual battlefields. RTP is also modifiable for different specific real-time application needs. RTP as a protocol does not guarantee QoS attributes like timeliness and in-order delivery and relies on the lower level layers of the network.

RTP is suitable for audio / audio+video conferencing applications with reception reports that are being send on different ports periodically by using RTP control protocol (RTCP). RTCP is responsible for sending reception feedbacks, transporting canonical identifiers for sender end points, and providing knowledge on the receivers in a multi-cast session in a session.

Since application with real-time attributes are sensitive against transmission errors of packets, it is a necessity to correct those errors in most situations. However, RTP itself does not provide any mechanism for error correction. In fact, continuous sequence number space of a synchronization source (SSRC) provides a platform to detect packet losses. Also, the fraction of lost information in RTCP report packets may also be useful in order to react to unsteady loss patterns. Either forward error correction or retransmission can be used to correct transmission errors by the help of RTP’s various features.

2.1 RTP

RTP is designed by IETF in order to transport real-time data between endpoints in a reliable way by using an unreliable medium. It is constructed by using application-level framing and the-end-to-end principle.

Application-level framing paradigm had been studied by Clark and Ten- nenhouse first, and states that only the application is in charge of trans-
Chapter 2. Background

port decision of data. In case of network problems, application-level framing gives application the privilege of reacting in a proper way such as recovering the lost packet or omitting the loss. Therefore, its usage is more compatible with UDP instead of TCP, which tends to hide such network problems with its built-in mechanisms.

The second key design principle is the end-to-end principle which suggest having smart end-points with dumb network hops. This helps simplifying the lower members of the protocol stack such as network layer and data link layer. In that case, applications require an additional effort to provide robustness on the transmission [?].

In general, an RTP stack gets a chunk of any kind of media data frame (video/audio) from an application, then constructs an RTP packet whose payload is the data received from the application. The RTP header of an RTP packet is visualized in Figure 2.1.

![RTP Header](image)

**Figure 2.1: RTP Header**

The first bit of the RTP header represents the version number which is defined as 2, respectively **P** stands for the existence of possible additional octet(s) at the end of the payload. **X** indicates the presence of an extension header to the standard RTP header. **CC**, the CSRC count, specifies the amount of contributing source identifiers in the RTP packet. **M field** is interpreted as the profile that is being used. **PT stands** for payload type, which is defined by 7 bits. The payload that is being carried in the RTP packet is interpreted by the application according to its payload type. In addition, a receiver can subscribe to a stream by looking at its payload type or simply ignore it, such as accepting video streams that are encoded with a specific codec.

The next 16 bits after payload type is reserved for the sequence number which is used for reordering the packets or detection of packet loss. Sequence number of a packet that is sent by an SSRC, is the sequence number of a previous packet sent by the same SSRC plus 1. Initial sequence number for an SSRC is randomly chosen, and thus every SSRC has their own sequence number space. Following the 16bits of sequence number, 32bits are reserved for the timestamps of the RTP packet. It states the sampling time of the first octet in the RTP packet, and used for synchronization. Synchronization source identifier is intended to be unique in an RTP session and chosen randomly. As it is clear from the name, it is a unique identifier for the sender device. Even though choosing the same SSRC randomly is unlikely, RFC3550 requires every RTP implementation to detect and solve such a
collision. The last 32 bits $\ast CC$, in case of extension bit is set to 0, before the payload in the RTP header is reserved for CSRC identifiers, if the payload of the RTP packet is a mixture of the data from different SSRCs, every contributing source is listed [?].

Detailed information on the RTP header fields, RTP mixers and translators can be found in RFC 3550.

2.2 RTCP

RTP control protocol (RTCP) is a monitoring mechanism for RTP, which conveys information on reception quality of RTP packets. RTCP should convey feedback reports for the RTP packets and a transport-level identifier called canonical name. These information is useful to adapt to changing conditions in a network.

There are 5 types of RTCP packets for different purposes. Sender report (SR) consists of transmission feedbacks and is constructed by an active sender in a session. Receiver report (RR), similar to SR, conveys reception feedbacks from those participants who are not active senders. Session description packets (SDES) consists of the canonical name (CNAME) identifier, user name, email, phone, geographic user location (LOC), application or tool name (TOOL), notice/status item, and a private extension for experimental purposes as a chunk for one single SSRC. BYE packets are used to signal a participant leaving a session. The last form of a RTCP packet is APP packets, which is defined by the specification for experimenting new features.

The overview given for RTCP protocol and its purposes with the packet types is sufficient in terms of the context of this thesis work. The detailed information about the packets and the fields they consist of, also the parsing and interpreting these packets, and actions to take can be obtained from RFC3550 [?].

2.3 Multiplexing Schemes

2.3.1 SSRC Multiplexing

In order to send original RTP and redundant (retransmission/fec) streams in the same session, a technique, called SSRC multiplexing, is used to differentiate those data streams by using different SSRC values for each.

SSRC is the distinguishing key to differentiate data streams coming from a source A, and also various number of statistics needs to be kept for a particular remote/local SSRC (RTCP Statistics). SSRC Multiplexing reduces number of port usage, by sending different streams in the same session, thus through the same end point.
2.3.2 Session Multiplexing

Session multiplexing is another way of separating different streams, by sending them in different sessions. For instance, [?] states that video and audio streams should not be sent in the same session. In RTP, session multiplexing is provided by using different end points (network address and port pairs) for distinct sessions.

Transmitting different streams in the same session cause ambiguity whenever encodings for the streams that are generated by the same SSRC are changed. In addition, using the same SSRC for different streams in the same session cause also sharing the sequence number space and a single timing space, which are intended to be different for different streams due to the different clock rates of the payloads and necessity of loss detection for different streams by using the sequence number of the packets. These problems can be overcame by using SSRC multiplexing, however, it may not suffice some cases. For instance, not using session multiplexing would prevent the use of different network traversals and parallel implementations for processing different media types. [?] also states that, RTP mixers would not be able to combine interleaved media streams with distinct clock rates. Thus, audio and video streams are proposed to be carried in different sessions.

In case of transmission of redundant data that is generated by a retransmission or fec scheme for a specific payload type, either SSRC or session multiplexing can be chosen for a unicast session, due to the fact that the problems mentioned above that SSRC multiplexing is vulnerable are not applicable. However, session multiplexing is a necessity in a multicast session due to the multiple incomplete retransmission requests from different receivers for the same sequence number in different streams [?]. In unicast sessions this can be taken under control.

2.4 Forward Error Correction in RTP

As mentioned earlier, forward error correction is another method of recovering packet losses and suitable for the cases where the delay that retransmission causes is beyond an acceptable value for a real-time application. In this thesis work, the focus is on payload-independent forward error correction schemes such as Parity and Reed-Solomon codes. Both of these schemes are block codes, thus they are applied on a fixed-size data block, group of packets in our case, in order to generate repair data.

In order to implement necessary mechanisms of forward error correction such as encoders, decoders for different block codes, and essential stack components, receiver and sender sides are considered separately. In general, receiver side is more complex than the sender side because of the packet loss detection and the reconstruction of the related encoding block, which is a combination of the received FEC packets and the original RTP packets.

FEC packets, regardless of the FEC scheme being used, are carried in-
side RTP packets. Therefore, every FEC packet has a regular RTP header as stated in Figure 2.1 with a dynamic payload type determined by the application. In addition, in case of using session multiplexing for stream differentiation, SSRC field of the fec packet is the same as the original packets. Since, fec packets are generated after a certain number of RTP packets, timestamp of a fec packet is set to the original media RTP clock at its encoding instant \[?\].

Applying forward correction in the RTP stack requires a buffer with a size that is suitable to store all originals packets received successfully and needed for recovery of the packet losses occurred. In this work, a FIFO approach is used to store outgoing RTP packets both in sender and receiver, however any kind of data structure might be suitable for such a container. The importance of the data structure for buffering RTP/FEC packets is emphasized in the discussion section. On the other hand, determining the source block size in a feasible way not to exceed bandwidth limitations and minimizing the residual packet loss, is another challenge. In the next section, Parity FEC and Reed-Solomon schemes are described in detail with their limitations.

2.4.1 Parity FEC

One of the most straightforward block code is the Parity code. The main principle is constructing a redundant bit stream by applying bitwise exclusive-or operation (XoR) on a source bit stream. Then, on the receiver side, if one packet in the source stream is lost, the lost packet can be recovered by applying the bitwise XOR operation on the successfully received packets and the FEC packet associated with the corresponding part of the bit stream.

Since adding a redundant bit stream consumes bandwidth, and bandwidth is a restrictive resource in networks, the challenge of forward error correction reveals itself as finding the most efficient block size and type to apply protection operation. In addition, which scheme to use should also be determined according to the needs to recover packet losses.

Parity FEC is based on XoR operation on different properties that are needed to reconstruct a packet such as length of a packet, payload type, timestamp, and the payload itself. In addition, FEC header requires some extra fields, as showed in Figure 2.2, corresponding 12 octets + 4 octets (optional).

In order to construct the Parity FEC packet, \(n\) number of packets must be buffered. Since the simple XoR operation on \(n\) number of packets is a fixed sized operation, the maximum packet size in the source block should be retrieved to determine the padding needed for each packet. Length of a packet in bytes is defined as the summation of the size of the payload, CSRC count multiplied by 4- since one CSRC is 4 bytes-, padding of the packet, and header extension, if there is any. Thus, length recovery is obtained by performing XoR of lengths of each packet in a source block. Similarly,
timestamp and payload type of the FEC header is calculated by applying XoR operation on the same fields of the original RTP packets.

<table>
<thead>
<tr>
<th>E</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>Recovery</th>
<th>SN</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.2: Generic FEC Header [?]  

The basic property of XoR operation that lies behind parity forward error correction is

\[ a_0 \oplus a_1 = b; b \oplus a_1 = a_0. \]

Thus, by using this property one missing symbol in the source block can be reconstructed in another XoR operation.

There are three different FEC policies depending on the order of original RTP packets that are used in a source block in order to create a Parity FEC packet. These are called as 1D Row, 1D Column and 2D matrix policies.

As shown in Figure 2.3(a), with 1D Row policy, \( n \) number of consecutive RTP packets are encoded according to the Parity FEC scheme. The receiver will have to wait until \((n - 1)\) number of packets in the encoded source block are received, plus the FEC packet to recover 1 lost packet at maximum. Thus, the delay is only introduced in the receiver side. In the worst case, if the first packet does not reach to the receiver, the receiver must wait until the next \((n - 1)\) packets, and the FEC packet to start the recovery operation. Moreover, the bandwidth overhead, which is the ratio of the size of FEC packets in bytes added to a source block to the size of the source block in bytes, is equal to

\[
\frac{\max_{0 \leq i < n} [s_i] + S^{FEC}}{\sum_{i=0}^{n-1} s_i + (n \times S^{RTP})}, \tag{2.1}
\]

where \( s_i \) is the payload size of a source packet, \( S^{FEC} \) is the header size of a FEC packet, and \( S^{RTP} \) is the RTP header size.

1D Column policy is very similar to 1D Row policy in terms of source block construction but the difference is the sequence numbers of the packets in the source block are not consecutive, but increasing at a certain rate. Thus, this policy requires shaping the source block as a matrix instead of a vector of RTP packets. Since this policy constructs a matrix of packets then generates FEC packets from those corresponding to one column of the matrix as shown in 2.3(b), single packet loss occurs in one column and burst losses occur in a row can be recovered. If \( n \times m \) matrix is chosen
to construct FEC packets from columns, the first FEC packet is generated after the \((n \times (m - 1)) + 1\)th packet and the last FEC packet is after \((n \times m)\)th. Therefore, in the worst case, the delay on the receiver side is the time required to receive \(n \times m\) packets plus \(n\) number of FEC packets. The bandwidth overhead is

\[
\sum_{i=0}^{m-1} \max_{0 \leq k < n} [s_{i+m \times k}] + S_{FEC}^{FEC} \sum_{i=0}^{(n \times m)-1} s_{i} + (n \times m \times S_{RTP})
\]

(2.2)

The last Parity FEC policy this thesis work covers is 2D matrix. In this policy, source block is constructed as in 1D Column policy, but generating redundancy packets also in a second dimension, horizontally. The major benefit of having the redundancy packets generated in the second dimension is the increment in the error correcting capability. In other words, 2D matrix policy eliminates the vulnerability in both 1D Row and Column policies as showed in Figure 2.3(c). For instance, in case of multiple packet losses in a row or a column might be recovered by a recursive recovery algorithm.

The main tradeoff between the error recovery rate and the bandwidth overhead reveals itself in three different policies of Parity FEC. The more protection requires generating more redundancy packets. Hence, the bandwidth overhead of 2D matrix policy is defined as:

\[
\frac{\sum_{i=0}^{m-1} \max_{k} [s_{i+m \times k}] + S_{FEC}^{FEC} + \sum_{i=0}^{n-1} \max_{j} [s_{i+n \times j}] + S_{FEC}^{FEC} \sum_{i=0}^{n \times m-1} s_{i} + (n \times m \times S_{RTP})}{n \times m}
\]

(2.3)

where \(0 \leq k < n\) and \(0 \leq j < m\).
Chapter 2. Background

2.4.2 Reed-Solomon Codes

Reed Solomon codes are some form of a BCH codes, designed to correct multiple random errors and being widely used in many applications such as digital audio discs, data transmission technologies, and deep space telecommunication systems [?].

RS codes are represented as RS(n,k) where \( n - k \) is the number of repair symbols being generated, and \( k \) is the number of source symbols. The generated repair symbols must be existed over the same field as the source symbols, thus all operations have to be performed in such a finite field. Galois Field, GF(256) in particular, is a good candidate for all operations to be performed in our RS encoder and decoder. Since the decoder at the receiver
Chapter 2. Background

side needs to wait until \( k \) number of packets are received successfully, there’s a latency introduced by using the RS codes for forward error correction \([7]\).

In addition, the bandwidth overhead in bytes is

\[
\frac{\left( n - k \right) \times (S^{\text{FEC}} + \max_{0 \leq i \leq k} [S^p] + l)}{(k \times S^{\text{RTP}}) + \sum_{i=0}^{k} s_i},
\]

where \( S^{\text{RTP}} \) is 8 bytes, \( l \) is the RS code of the lengths of the packets in the related source block, \( S^p \) is the original packet size, and \( s \) is the payload size of packets in the source block.

Generating RS codes is based on evaluating \( m \) polynomial functions of degree \( n \) at \( m \) number of source symbols, which results in \( n - m \) encoding symbols, where \( n > m \) and first \( m \) number of generated encoding symbols are equal to source symbols. Thus, if coefficients of \( m \) polynomial functions of degree \( n \) replaced in a matrix, an \( mxn \) matrix is obtained, and the operation is equal to a matrix multiplication with a vector constructed from source symbols, which can be mathematically represented as

\[
y = A \cdot x.
\]

In case of a any number of lost source symbols smaller or equal than \( n - m \), any lost symbol can be recovered with the following operation

\[
x_i = B^{-1}[i][m] \cdot y,
\]

where \( B \) is the inverse of extracted \( m \) rows of \( A \) corresponding to the respected order of the received decoding symbols, \( i \) is the missing symbol’s index. Hence, a special matrix that keeps non-singular property in case of any \( (n - m) \) rows are extracted is needed to accomplish encoding and decoding RS codes.

2.4.2.1 Galois Field Arithmetic

Field is an algebraic system, where the basic operations such as addition, multiplication, subtraction, and division, is defined within the field and results within the field. In addition, for a set of elements with the operations defined to be a field, an identity element for addition operation should be defined and denoted by 0, multiplicative identity should be defined and equal to 1, and multiplication must be distributive over addition.

Galois field (GF) is a finite field, and carries significance in many areas such as coding theory, digital data transmission, cryptography, number theory, etc..., and named after Évariste Galois due to his contributions to group theory and Galois theory.

GF(2) is a binary field where the elements are either "1" or "0" and multiplication and addition operations are defined under modulo 2. For any prime number \( p \), it is also possible to show that a set of elements \([0...p - 1]\)
constructs a field with multiplication and addition operations defined in modulo $p$. Therefore, for any positive integer $m$, it is also possible to extend a field $GF(p)$, which has $p$ elements, to $p^m$ elements, while keeping the properties of being a field. These fields are called extension fields and denoted as $GF(p^m)$. The statements mentioned here are straightforward to prove by construction addition and multiplication tables among $p$ elements over modulo $p$ as stated in [?].

In Reed Solomon codes, all operations are done in a field $GF(q)$, where $q = 2^m$ and $m$ is the number of bits in a source symbol to be encoded. Due to the fact that 8 bit symbols are used in the current RTP stack implementation, $GF(2^8)$ is used for all operations. $GF(2^8)$ can be constructed by a primitive polynomial $p(x)$ which only has trivial divisors, and given as $p(x) = 1 + x^2 + x^3 + x^4 + x^8$ in [?] by using $\beta = x' \mod p(x)$ where $0 \leq i \leq 2^m - 1$ and $\beta$ is a root of $p(x)$.

For practical reasons, multiplication of $a$ and $b$ in $GF(2^8)$ can be performed as logarithmic operations in base 2 by using the following property:

$$ab = \log_2^{-1}(\log_2 a + \log_2 b),$$

(2.7)

Thus, log tables and their inverses can be precomputed since they are being used vast amount of times.

### 2.4.2.2 Vandermonde Matrices

Vandermonde matrix is a special formed matrix name after Alexandre-Théophile Vandermonde, where each row is constructed by an element scaled with a ratio until you reach an order $n$, as showed in 2.8.

$$V = \begin{bmatrix}
1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{n-1} \\
1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{n-1} \\
1 & \alpha_3 & \alpha_3^2 & \ldots & \alpha_3^{n-1} \\
1 & \alpha_4 & \alpha_4^2 & \ldots & \alpha_4^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \alpha_m & \alpha_m^2 & \ldots & \alpha_m^{n-1}
\end{bmatrix} \quad (2.8)$$

Constructing a Vandermonde matrix with $m$ distinct values results in a non-singular matrix, from which an RS decoder is benefited by constructing a square matrix $(m \times m)$ from original matrix’s $(m \times n)$ columns (or rows in case the matrix is transposed) corresponding to the first successfully received $m$ symbols’ order. Since the determinant of a square Vandermonde matrix is

$$V_n = \prod_{1 \leq i < j \leq n} (\alpha_j - \alpha_i), \quad (2.9)$$

and there is not such a case as $\alpha_j = \alpha_i$ where $i \neq j$ because of $\alpha$’s linearly independency. By this definition, it is obvious to say that, any chosen $n$
rows of the matrix \( V \) is subject to construct a non-singular matrix. There is one drawback of Vandermonde matrices called, ill-conditioned. Being an ill-conditioned matrix states that condition number, which gives an estimation on the precision loss, is large. Knowing \( V \)'s coefficients are known up to a given precision will generate huge variation in \( V^{-1} \), thus in the solution of \( x = V^{-1} \cdot y \). However, since we work in finite fields, we won’t suffer from this drawback.

### 2.4.2.3 Gauss-Jordan Elimination for Vandermonde Matrix Inversion

In order to invert a square Vandermonde matrix \( V \), a fairly fast algorithm is needed. Gauss-Jordan Elimination technique is chosen to achieve this aim even though it is 3 times slower than LU Decomposition method in solving linear equations [?] due to its simplicity and suitability for finding the inverse of an invertible square matrix.

\[
A \cdot A^{-1} = I \tag{2.10}
\]

Gauss-Jordan Elimination technique is used with full pivoting in our implementation not to stumble upon 0 divisions in where the diagonal element becomes zero due to the previous row reductions as stated in [?].

The main principle is transforming matrix \( A \) to reduced row echelon form by applying basic row and column operations by reflecting them on the same sized identity matrix. The set of operations on the applied matrix \( A \), turns identity matrix into \( A^{-1} \). In addition, all arithmetic operations must be applied in \( \text{GF}(2^m) \).

### 2.5 System Description

The RTP Stack, into which forward error correction techniques are subject to be implemented, consists of different modules that works in the chain of responsibility pattern. All modules have two ways of dealing with data as receiving and sending. In general terms, an incoming RTP packet through the local endpoints is processed for gathering statistics then passed to the next module. The modules in order hand over the incoming packet and apply their responsibilities on it. The payload is extracted in some module before the sequence reaches to the application. In the meanwhile, appropriate RTCP reports are sent in the specified interval. The sending process is the backwards operation of the receiving process.

In order to be able to send redundancy packets (FEC packets) in a second session, a second session with different port pairs is created with a module that handle FEC packet generation. In addition, the current session model is extended with a pre transport module, which handles the recovery of lost packets by using the related RTP and FEC packets. Upper layers in the
Chapter 2. Background

stack would treat the recovered packet as it is received normally, however, statistics for the related SSRC should not be affected by the recovery process. In other words, even though a lost packet is generated by using FEC, the sequence number of the related packet should still be counted as missing in the statistics. In any other case, it may affect the reception reports and loss rates for the related SSRC.

Every session has a receiver thread, thus session multiplexing requires thread synchronization. Maintaining the chain of responsibility pattern underlying the stack implementation requires the correct sequence of packet passing between the modules in different sessions. Whenever an incoming RTP packet arrives through the local endpoint A that belongs to the session 1, it is treated as described above. RTP Packets that carries FEC payloads, arrive to the session 2 through the local point B. After it’s been passed among the various stack modules, the FEC payload module passes it to the session 1’s module that is related to FEC receiving operations. The overview of the module replacement can be seen in Figure 3.1
2.5.1 Test Application

The test application, which is linked to the RTP stack with FEC modules, simply simulates a receiver/sender for unicast dummy data transmission with tuneable packet size and packet generation frequency. In order to mimic RTP packets carrying video data with payload type H.264, a packet generation algorithm is implemented. Stack’s configuration classes are disclosed to the application through an application programming interface (API), which gives access to various statistics such as number of packets transmitted, number of packets lost, number of packets recovered, number of redundancy packets sent, etc., to the application.

In addition, FEC and Retransmission plugins are configurable by the application through APIs, according to the parameters parsed in the SDP. Snapshot of RTP and FEC packets generated by the test application and sent through session multiplexing is shown in Figure 2.5.

Figure 2.5: Wireshark capture of the packets generated by the test application.
Chapter 3

Methodology

The main focus on using forward error correction in terms of quantitative values is obtaining residual packet losses that is acceptable, where round trip times are high (> 200 ms). To simulate these packet losses, arbitrary loss patterns in an erasure channel needs to be created.
Chapter 3. Methodology

3.1 Implementation

In order to implement generic forward error correction modules in an RTP stack, the structure of the stack implementation with respect to the general design pattern is used needs to be carefully studied. The Ericsson’s RTP stack - called as ”Yet another RTP stack” (YARS)- is a typical example of chain of responsibility, and written in C++, by using boost library to maintain portability.

Due to the fact that session multiplexing is the chosen method, a second session as a child session of the main session needs to be created and owned by the main session. In fact, the application is unaware of the presence of a second session which is used to transmit FEC streams. In addition, the complete RTP stack implementation should be capable of handling both receiving and sending FEC streams, which are configurable by choice from the application.

FecPayloadModule and FecPreTransportModule are two generic FEC modules that are needed to be implemented, and have distinct responsibilities with respect to the role. FecPreTransportModule, which is placed in the main session right before the transport module, is where the decoding and recovery logics are implemented in the receiver side. However, on the sender side it only functions as a proxy to forward the copy of an RTP packet that is sent by the application to the FecPayloadModule. All source block construction, encoding, and FEC packet generation is FecPayloadModule’s responsibility, which is placed in the child session.

On the receiver side, the module’s responsibilities are reversed. FecPayloadModule is responsible of forwarding FEC packets to the main session’s FecPreTransportModule and encode block construction, decoding operation, and packet loss recovery is implemented in the FecPreTransportModule. Module replacement and the flow directions are shown in Figure 3.1.

![RTP Stack with FEC modules](image)

Figure 3.1: RTP Stack with FEC modules.
Chapter 3. Methodology

3.1.1 Parity FEC Encoder and Decoder

In order to maintain scalability for further algorithm implementation, FEC modules are designed as generic as possible, whereas specialized algorithms for decoding and encoding source/encode blocks are specially treated. As a brief explanation, all encoders inherits an abstract base class called \textit{FecEncoder}, and all decoders are derived from another abstract base class called \textit{FecDecoder}. The purpose of implementing such a class design is to obtain a strategy pattern where all encoders and decoders, the current implemented ones and the ones might be implemented in the future, are seen as the same by generic FEC modules. This is intended to give flexibility in terms of expanding forward error correction support in the RTP stack. Generic class design for encoders and decoders are given in Figure 3.2.

![Class diagram for FEC encoders and decoders.](image)

Figure 3.2: Class diagram for FEC encoders and decoders.

Parity FEC operation is a computationally cheap operation, and has a straightforward implementation. In general, any encoder is invoked when the RTP packet buffer is reached to the source block extent. Since \textit{FecPayloadPlugin} is not aware of which encoder is supposed to be invoked, after retrieving the correct encoder for the related stream, a virtual method is called to initiate the encoding operation.

Packet header construction for Parity FEC and applying protection operation on packet’s payload in a source block needs to be done as fast as possible, not to cause an extra latency for the receiver. Since the payload size’s in a source block can vary, all the short payload data needs to be padded with 0s before the protection operation starts. To take care of these details and not to make unnecessary traversals in the packet buffer, the FEC packet header is constructed in the first loop whereas payloads and their sizes are buffered externally. After the header construction and protection operation on the payloads of RTP packets that are involved, constructed FEC packet is wrapped as an RTP packet with a timestamp and the same SSRC as the RTP packet’s. As [?] states, the timestamp of the generated FEC packet is set to the time instant at which it is being sent over. For such a calculation, the elapsed time since the last RTP transmission of the SSRC and clock rate of the original media need to be gathered. Section 2.4.1 gives
Chapter 3. Methodology

an insight on how the header fields are obtained.

At the decoder side for Parity FEC algorithm and its three schemes, another challenge exists, determining the number of packet losses in a source block and recovery attempt. Among three schemes, 2D scheme is the most critical one in terms of complexity at the receiver side, because of the fact that a successful recovery has a possibility to lead further recoveries inside the source block. In general, when a FEC packet or RTP packet is received, a recovery attempt must be invoked for instant correction. Since, the recursive recovery is needed, FecDecoders are designed to call a function on FecPreTransportModule to convey the recovered packet instantly. In case of further recovery is needed, a copy of the recovered RTP packet is restored in the buffer.

RTP packet buffer sizes for both decoder and encoder side is needed to be specified carefully, since they have an important role on encoding/decoding speeds and memory usage of the whole stack. Thus, it is configured to source block size \( \times 1.5 \), in order to have enough room for packet reordering and in case of encoding/decoding time exceeds the packet receiving rate. Another tuning is done to make recovered packets bypass the statistics module of the main session. Instead, recovered packet statistics are recorded separately and made accessible through the FEC config API.

3.1.2 Reed Solomon Encoder and Decoder

ReedSolomon encoder and decoder are very similar to Parity FEC encoder and decoder since they are also derived from FecEncoder and FecDecoder classes. The virtual methods are overwritten in order to apply ReedSolomon operations for repair symbol generation. In contrast to Parity FEC, ReedSolomon codes are computationally expensive. In order to reduce the complexity in run-time, frequently related members are precomputed such as Vandermonde matrix with size \( n \times k \), square subset of this Vandermonde matrix, with size \( k \times k \), and Galois field library, which also precomputes log and antilog tables offline. For Galois Field operations, an open source library called "Galois field arithmetic library" written by Arash Partow is used with minor adjustments to the current system [7].

Figure 3.3 shows how the source block is constructed, and on which data the RS codes are generated. It also gives an overview of where the original length of the application data units (ADUs) should be placed, and the padding to obtain a even sized block. After constructing the source block, the RS code algorithm is performed exactly \( n \) number of times where \( n \) is the length of the longest RTP packet in the source block. Since such an high number of the same operation results a high complexity itself, the precomputed matrix are used as the generator matrix by reducing the memory access times.

An extra complexity that ReedSolomon decoder adds is the error location detection that is being needed to extract corresponding rows from the
Chapter 3. Methodology

Vandermonde matrix at the receiver side. In order to minimize the search space for the locations of loss packets, a specialized buffer is used to store incoming RTP packets, which is keyed with the sender SSRC and base sequence number, which can be obtained from the FEC header. Encoding block is constructed in a similar way to the source block, however, the first $k - n_{\text{lost}}$ ADUs, where $n_{\text{lost}}$ refers number of lost packets, are belong to the original RTP packets, and the rest belong to the received repair packets.

The recovery operation in ReedSolomon codes is one time operation and does not need to be repeated recursively. In addition, possibly recovery check whenever an RTP packet arrives is unnecessary, due to the assumption of receiving repair packets later than the original RTP packets in the source block. Thus, decoding operation causes a latency as much as the time is needed to receive $k - n_{\text{lost}}$ original RTP packets + $n_{\text{lost}}$ repair FEC packets.

![Figure 3.3: Source block generation for ReedSolomon encoder.](image)

### 3.1.3 Fec Config API

Fec Config API is the interface of the application to configure FEC streams for original RTP streams. The main purpose is to give application the control of accepting, ignoring, sending FEC streams for the specified RTP streams. It also gives the flexibility to the application to switch between implemented algorithms, that are agreed upon by both parties in an outbound signaled session description protocol (SDP) message. The implemented API consist of functionality to register the agreed payload type for FEC packets to the RTP stack, and make FEC modules ready to generate and receive FEC packets according to the given algorithm type and source block sizes.
3.2 Network Evaluation

Packet loss is an event that occurs when an IP packet does not reach its destination because of various reasons. The network congestion, where router buffers are filled up with packets, late packet arrivals, also called as jitter, or data corruption might be reasons for packets to be dropped in networks. Packet losses have significant effects on real-time communication systems over lossy channels, especially where UDP is used as transport protocol, and reliable packet transmission is not guaranteed.

According to the experiments on packet loss during a multimedia conference with multicast/unicast sessions in the Internet, done by Yajnik, Moon, Kurose, and Towsley, the length of error bursts are small [?]. It is stated that their traces showed no longer than 4 consequent packet losses. In addition, Figure 3.4, shows the distribution of burst error lengths, obtained from the trace where a packet stream is transmitted from Amherst, Massachusetts to Swedish Institute of Computer Science (SICS), Stockholm at sampling interval 160ms. Moreover, the traces on which their measurements are, gives a variation of packet loss rates between 1.4% and 11% (11% packet loss is obtained in only one of the traces, the others indicate less than 5.3%).

![Figure 3.4: Burst loss distribution for the trace between Amherst, Massachusetts - SICS, Stockholm with loss rate 1.9%.](image)

Another study on residential broadband performance across various ISPs in the United States by Sundaresan, de Donato, Feamster, et. al. [?] obtained small packet loss rates with high variances. In addition, it is stated that bursty packet losses are more likely compared to single losses. Table 3.1 shows their measurements across various ISPs within United States.
Chapter 3. Methodology

<table>
<thead>
<tr>
<th>ISP</th>
<th>Loss Avg%</th>
<th>Std. dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT&amp;T</td>
<td>0.58%</td>
<td>3.59</td>
</tr>
<tr>
<td>Comcast</td>
<td>0.27%</td>
<td>2.79</td>
</tr>
<tr>
<td>TimeWarner</td>
<td>0.33%</td>
<td>3.09</td>
</tr>
<tr>
<td>Verizon</td>
<td>0.51%</td>
<td>4.07</td>
</tr>
<tr>
<td>Charter</td>
<td>0.43%</td>
<td>3.29</td>
</tr>
<tr>
<td>Cox</td>
<td>1.11%</td>
<td>8.35</td>
</tr>
<tr>
<td>Qwest</td>
<td>0.33%</td>
<td>3.38</td>
</tr>
<tr>
<td>Cablevision</td>
<td>0.33%</td>
<td>3.14</td>
</tr>
</tbody>
</table>

Table 3.1: Loss rate measurements with variance across various ISPs. [?]

With respect to the suggested bursty packet losses stated above, it is obvious that consequent packet losses would degrade video quality. The study done by, Boulos, Parrein, Le Callet and Hands states the distortion caused by bursty losses. The effects of burst losses depends on the cases. For instance, when the encoding bit-rate is low and one single frame can fit in one RTP packet, it is expected that shorter bursts would cause significant distortion. On the contrary, the high bit-rate frames would fit into multiple RTP packets, therefore longer bursts would create the same effects. In their measurements, 8 video sequence are used with different characteristics such as high/low motion, textured regions, still images, scene cut etc. encoded in H.264/AVC and transmitted via RTP over UDP. All sequences are in standard definition (720 x 576). The loss patterns are being used varies in packet loss ratio between %0.2 to %8.9 and burst lengths between 3 and 40. Their findings indicates that the mean opinion score (MOS) is depended not only the packet loss ratio but also the number of group of pictures (GOP) effected. For low motioned videos, such as in a video conference application, the QoE is effected worse in case of a single long burst than multiple losses [?].

3.2.1 Generating Packet Loss Models

In order to simulate burst packet loss, we use Gilbert-Elliott model, which is a 2 state Markov chain as shown in Figure 3.5. G and B stands for two states of the network, Good and Bad respectively. Gilbert’s model suggests having the Good state as error-free and introducing errors in the Bad state with an error probability [?]. Elliott contributed Gilbert’s method by introducing errors also in the Good state. To keep the simplicity of our error channel modeling by obtaining resemblance to the real cases, we use the simplified Gilbert method, by considering the good state as error-free and the bad state as error-prone.

The probability of a network switching from the Good state to the Bad state at a time instant $t$ is $p$, from the Bad state to the Good state is $r$. 
Chapter 3. Methodology

Hence, the probability of a network maintaining its Good state is $1 - p$, and its Bad state is $1 - r$ at the time instant $t + 1$. The transition matrix can be derived from Equation 3.1.

$$P = \begin{pmatrix} 1 - p & p \\ r & 1 - r \end{pmatrix}$$  \hspace{1cm} (3.1)

The value $r$ can be considered as the burst controller since it has an effect on staying in the bad state. Thus, the probability of a burst with length $n$ occurring is be calculated as:

$$p \times (1 - r)^{n-1}$$  \hspace{1cm} (3.2)

And the theoretical error rate is

$$1 - \frac{r}{r + p}.$$  \hspace{1cm} (3.3)

The simply Gilbert method is implemented in Matlab in order to generate packet loss traces of 40,960 packets, intended to use as a packet loss configuration in the network simulator software NetDisturb. The reason of choosing the number 40,960 is NetDisturb’s maximum limit on loss values extracted from a file input. $p$ and $r$ values are obtained through the experiments with the implemented software to generate desired results which are the variations of the real network measurements. Table 3.2 shows theoretical packet loss rates as well as $p$ and $r$ values used to generate packet loss patterns that has corresponding generated packet loss rates and maximum burst lengths, which are used in the experiments.

In order to give a more clear view on the bursty loss distribution in the generated models, Table 3.3 shows the number of burst loss occurrences that exist in each generated model.

Since it is stated earlier that group of consecutive losses may place in the same source block of a FEC encoder, thus they create the same effect of
Chapter 3. Methodology

<table>
<thead>
<tr>
<th>model #</th>
<th>p</th>
<th>r</th>
<th>max_burst</th>
<th>theoretical loss</th>
<th>generated loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0010</td>
<td>0.7934</td>
<td>2</td>
<td>0.1258%</td>
<td>0.1196%</td>
</tr>
<tr>
<td>2</td>
<td>0.0010</td>
<td>0.7934</td>
<td>4</td>
<td>0.1258%</td>
<td>0.1171%</td>
</tr>
<tr>
<td>3</td>
<td>0.0009</td>
<td>0.5000</td>
<td>5</td>
<td>0.1796%</td>
<td>0.1684%</td>
</tr>
<tr>
<td>4</td>
<td>0.0135</td>
<td>0.8930</td>
<td>3</td>
<td>1.4892%</td>
<td>1.5014%</td>
</tr>
<tr>
<td>5</td>
<td>0.0135</td>
<td>0.8930</td>
<td>4</td>
<td>1.4892%</td>
<td>1.5209%</td>
</tr>
<tr>
<td>6</td>
<td>0.0701</td>
<td>0.8700</td>
<td>4</td>
<td>7.4566%</td>
<td>7.3022%</td>
</tr>
<tr>
<td>7</td>
<td>0.0701</td>
<td>0.8700</td>
<td>5</td>
<td>7.4566%</td>
<td>7.3413%</td>
</tr>
<tr>
<td>8</td>
<td>0.0591</td>
<td>0.7900</td>
<td>6</td>
<td>6.9603%</td>
<td>6.9433%</td>
</tr>
</tbody>
</table>

Table 3.2: Generated Simple Gilbert Model Parameters

<table>
<thead>
<tr>
<th>model #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>478</td>
<td>54</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>484</td>
<td>54</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2181</td>
<td>274</td>
<td>37</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2182</td>
<td>285</td>
<td>31</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1657</td>
<td>361</td>
<td>87</td>
<td>20</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.3: Burst distribution for generated Simple Gilbert models. Trace size is 40,960.

A one long bursty loss, visualization of the bursty loss distance in a sample loss model is given in Figure 3.6. Every packet loss is indicated with a white line on a grey plane, thus close packet losses constructs a larger white area which may results in the effect of a large consecutive packet loss.

Figure 3.6: Burst distribution of the trace generated by implemented Gilbert method. $p = 0.01923$ and $r = 0.80123$ where max_burst_length = 4.
Chapter 3. Methodology

3.2.2 Use of NetDisturb

NetDisturb is a network simulator software, which provides various ways to apply impairment on packet flows, and used in this thesis to obtain performances of implemented FEC algorithms with suitable parameters. The configuration for the testing environment is as shown in Figure 3.7.

![NetDisturb configuration overview for packet loss simulation.](image)

Figure 3.7: NetDisturb configuration overview for packet loss simulation.

The test application on top of the RTP stack with FEC enabled runs on Sender and Receiver where Sender generates dummy RTP packets, which have random bit sequences as payload, every 30-90ms as 1-6 bursts, and last packet’s mark bit is set. The burst groups shares the same timestamp with H.264 payload type, however the payload data is dummy and 1200 bytes + RPT header (min 12 bytes) + UDP header (8 bytes) + IP header (20 bytes) sums up to 1240 bytes. Sender’s FEC module is also responsible of generating the FEC stream, and sending it over the specified UDP port. In the meanwhile, NetDisturb applies packet loss impairments on the stream that is being transmitted to the Receiver. Receiver is only responsible of receiving RTP and FEC streams, recording statistics, and initiating the recovery process when a packet loss occurs and the recovery is possible.

Applying packet loss patterns that is generated by the Matlab program by implementing simple Gilbert loss model, and tweaking $p$ and $q$ variables, is fairly straightforward. NetDisturb has a packet loss law window where a file with semicolon separated values can be given as input. Maximum number of values being read from a file input by NetDisturb is stated as 40,960 in its documentation, and these values corresponding to a packet being lost or successfully forwarded according to the given threshold. Hence, 40,960 values are generated in each file to be compatible with NetDisturb. NetDisturb’s client interface is shown in Figure 3.8.
Chapter 3. Methodology

Figure 3.8: NetDisturb client window during the simulation showing the throughput and the packet loss at the sampled time.

There are 16 different packet streams that can be filtered and impaired. In each streams view, common counters for the received/sent packets through two different NICs are updated, and delay/jitter, content impairment laws can be defined or adjusted. There’s also a detailed logging option that gives a text output on the operations being done by NetDisturb. After defining packet loss laws with the models that are generated, and the NetDisturb’s function being carefully tested by a comparison between the test application’s counters and NetDisturb’s counters on received packets, necessary filters are applied to isolate the lossy channel behavior on both the RTP packets and the FEC packets being transmitted from sender A to sender B.
Chapter 4

Results

The limitation of retransmission as being unsuitable where the round trip times are higher than 200 ms was previously studied at Ericsson Research. The results will be listed here aims to find an alternative solution to retransmission when round trip times are higher between end points. Moreover, overheads that FEC may brings had to be studied and clearly stated to determine its suitability.

During the tests, the test application linked with FEC implemented RTP stack is configured to send 100.000 RTP packets with 968.75 kbit/s throughput (100 packet/s) where the payload is 1200 bytes, and the loss patterns in Table 3.2 are applied on both FEC and RTP streams in an isolated behavior in each run. The algorithms are chosen according to their error capability and the match with the loss pattern being performed. Thus, Table 4.1 shows the loss pattern applied, the FEC algorithm in use, the chosen source block sizes, and the residual packet loss ratios as well as bandwidth overhead revealed.
Chapter 4. Results

<table>
<thead>
<tr>
<th>MODEL</th>
<th>MB</th>
<th>FA</th>
<th>SBZ</th>
<th>RPL</th>
<th>BO</th>
<th>GL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1DParityRow</td>
<td>10 x 1</td>
<td>0.04%</td>
<td>10.17%</td>
<td>0.1196%</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1DParityColumn</td>
<td>10 x 1</td>
<td>0.007%</td>
<td>10.17%</td>
<td>0.1196%</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1DParityColumn</td>
<td>8 x 5</td>
<td>0.018%</td>
<td>12.72%</td>
<td>0.1171%</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2DParity</td>
<td>8 x 5</td>
<td>0%</td>
<td>33.07%</td>
<td>0.1171%</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>ReedSolomon(32,24)</td>
<td>24</td>
<td>0.0%</td>
<td>33.60%</td>
<td>0.1684%</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1DRow</td>
<td>5 x 1</td>
<td>0.367%</td>
<td>20.35%</td>
<td>1.5014%</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2DParity</td>
<td>8 x 5</td>
<td>0.006%</td>
<td>33.07%</td>
<td>1.5014%</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>ReedSolomon(32,24)</td>
<td>24</td>
<td>0.0%</td>
<td>33.60%</td>
<td>1.5014%</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>ReedSolomon(28,22)</td>
<td>22</td>
<td>0.0%</td>
<td>27.49%</td>
<td>1.5209%</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>ReedSolomon(25,20)</td>
<td>20</td>
<td>0.312%</td>
<td>25.20%</td>
<td>7.3022%</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>ReedSolomon(27,20)</td>
<td>20</td>
<td>0.025%</td>
<td>35.28%</td>
<td>7.3022%</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1DParityColumn</td>
<td>4 x 6</td>
<td>2%</td>
<td>25.44%</td>
<td>7.3413%</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2DParity</td>
<td>5 x 6</td>
<td>0.15%</td>
<td>37.31%</td>
<td>7.3413%</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>ReedSolomon(32,24)</td>
<td>24</td>
<td>0.123%</td>
<td>33.60%</td>
<td>6.9433%</td>
</tr>
</tbody>
</table>

Table 4.1: Simulation results with the trace length 100K.

**MODEL**: Generated packet loss model from Table 3.2.

**MB**: Maximum number of burst loss in the model tested.

**FA**: FEC Algorithm.

**SBZ**: Source block size.

**RPL**: Residual packet loss.

**BO**: Bandwidth overhead calculated with the given formulas in Section 2.4.1 and 2.4.2.

**GL**: Generated loss ratio given in Table 3.2.
Figure 4.1 shows all calculated worst-case performance of retransmission where the round trip time upper limit is set to 200 ms. Until 200 ms is passed since the discovery of a packet loss, packet retransmission can occur multiple times in case of retransmitted packets are also lost. Worst-case scenarios implies the possibility of retransmitted packets are also lost with respect to the given packet loss values on the channel.

Figure 4.1: Retransmission worst-case performance.
Chapter 4. Results

Figure 4.2 gives a clear view of the performance comparison between retransmission and FEC algorithms that are tested where the round trip time is between 99.1 ms and 200 ms. Since there is enough time for only one retransmission, the probability of losing the original RTP packet and the retransmitted packet is calculated for retransmission’s worst-case performance. FEC performance measurements, which are independent from round trip times, are based on the trial runs. Hence, the FEC algorithms that resides under the blue line performs better than retransmission. However, bandwidth overhead of FEC algorithms as showed in Table 4.1, are always higher than retransmission’s bandwidth overhead. Hence, this extra bandwidth consumption needs to be taken into consideration according to the application needs.

Figure 4.2: Retransmission worst case performance when RTT = [99.1-200] ms vs. FEC algorithms.

Figure 4.3: Detailed look of Figure 4.2
Chapter 4. Results

Figure 4.4 also draws the similar outcomes but with limited selection of suitable FEC algorithms. The blue line represents the worst case performance where the RTT is between 65.3 and 99 ms. Since retransmission of a lost packet can occur maximum 2 times, worst case probabilities of residual packet losses are lower. As we already know that retransmission works perfect under low values of round trip times, worst case scenarios for lower RTTs does not take place in this document, due to the performances of tested FEC algorithms.

![Figure 4.4: Retransmission worst case performance when RTT = [65.3-99.0] ms vs. FEC algorithms.](image)

Figure 4.5: Detailed look of Figure 4.4
In conclusion, since forward error correction schemes do not depend on round trip times, it is expected to reach same residual packet losses under the experimented packet loss patterns. The loss patterns are chosen from the ISP measurements as stated in Section 3.2, however in case of a presence of higher packet loss rates and/or longer bursts in the packet loss pattern, source block size could be decreased to cope with more errors, at the expense of bandwidth overhead and encoding/decoding complexity.
Chapter 5

Discussion

In general, forward error correction technique to recover packet losses in the RTP level, is not as straightforward as retransmission in many aspects. FEC addresses the problem of insufficient retransmission performance under long round trip times. However, obtaining the proper algorithm according to the packet loss pattern, and the complexity of the algorithm as well as its implementation should be carefully treated. With respect to the simulations performed, it can be said that regardless the algorithm is being used, the buffer sizes affect the algorithms complexity directly.

Even though the burst length seems to be the delimiter of the algorithm choice in terms of error correcting capability, it is also obtained that the distance between bursts is the significant parameter that gives us an estimation on how much loss is likely to occur while transmission a source block. For instance, close enough 4 length burst losses might occur in the same block of 24 packets that are protected by an RS code (32,24). On the other hand, it is hard to find optimum parameters and the algorithm, which gives minimum overhead on bandwidth and minimum residual packet loss ratio for all packet loss patterns.

In addition, recovery delays that the chosen algorithms cause should be considered in a production implementation. As mentioned earlier, Parity FEC algorithm is fairly less complex compared to ReedSolomon codes, therefore the complexity is not the main concern. However, the high complexity of Reed Solomon codes yields performance problems mostly on the receiver side where the application’s sender thread is both passing original RTP packets through the stack modules and generating FEC packets at the same time. As [?] suggests, Fast Fourier Transform (FFT) can be used to have a logarithmic performance both on RS decoder and encoder due to the fact that Vandermonde matrix’s similarity to DFT matrix. Since the multiplication of the source symbol vector with Vandermonde matrix is a multi-point evaluation problem, and Vandermonde matrix has symmetry in itself, every point evaluation discloses the parts of the remaining operations.
However, upon the experiments with FFT, it is seen that a full sized Vandermonde Matrix that consists of elements from GF(256) needs to be created in order to have the symmetry in itself. Therefore, an RS(n,k), where $n = 255$ is required to be used.

In general, [?] offers a solution to overcome complexity problem of Reed-Solomon erasure codes. Tornado codes use different approach to construct equations from the source data symbols. The average number of variables in equations are relatively small compared to Reed-Solomon codes, thus resulting in faster encoding and decoding times while sacrificing the number of packets needed to recover possible losses, by a factor $e$.

In Figure 4.2, it is stated that test runs for most of the FEC algorithms under various packet loss patterns resulted better residual packet losses than the retransmissions worst case scenario. In addition, some of these FEC schemes remain preferable over retransmission even RTT is down to 65 ms as presented in Figure 4.4. However, predictions for packet loss pattern on the channel and round trip time between end points need to be measured in order to determine an optimal FEC scheme for packet recovery. Moreover, FEC is the only option where RTT is over 200 ms due to the application’s restriction on the waiting time for a lost packet.

Bypassing the negative effects of high round trip times, and feedback implosion in retransmission by using forward error correction schemes may yield a focus also on other type of applications require reliable bulk data distribution. The experiments are done in unicast sessions during this thesis work, however Byers, Luby, Mitzenmacher, and Rege are focused on reliable bulk data distribution to numerous receivers by designing a framework including multicasting and forward error correction [?].
Chapter 6

Future Work

In order to obtain performance drawbacks that FEC schemes generates, and to measure its effect on the audio/video quality, a real application which encodes/decodes real media data should be used as a future work. Thus, fine tuning in FEC implementation can be done to reduce those drawbacks. On the other hand, it is obligatory to have multi-threaded design on the sender side as it is achieved by session multiplexing that uses distinct threads on the receiver side.

Since network conditions in terms of packet losses may not be stable and vary during a RTP session, the initial handshake for the protection strength of FEC scheme may consume unnecessary overhead or result in high residual packet loss rates. In order to prevent this, a dynamic switching scheme can be used to adjust protection level or set a different FEC scheme. It is important to state that all the FEC schemes that could be used during an RTP session needs to be in the SDP handshake before early media starts. Thus, both receiver and sender applications must be configured to expect the type of FEC packets that are acknowledged during the session initiation.

The most important problem with the current implementation is using one single application thread for both sessions. Since, FEC encoders are complex enough to decrease the application’s bit rate, FEC encoders should work on a different thread that does not block the application from sending RTP packets as frequent as its bit-rate requires. In order to achieve multi-threading for different session on the sender side, it is obtained from the experiments that some form of a non-blocking queue needs to be used to buffer outgoing RTP packets. Since the writer of the queue would be the application thread, it should not be blocked by the FEC thread that reads the queue whenever the source block size is reached. However, implementing such a non-blocked queue is not straightforward.
Video conferencing applications have reached worldwide usage in recent years by the help of the improvements in network infrastructures for public services. Media data covers a significant ratio of data traffic over IP networks. However, it is challenging to ensure a decent quality of service (QoS) on public networks in terms of video and audio quality.

The main factor that may cause degradation in media playback quality is packet losses. There are various techniques available to conceal packet losses in lossy channels. According to the application needs and channel characteristics such as loss patterns and round trip times, retransmission or forward error correction techniques may be applied at application level. These two techniques have different challenges which lead to tradeoffs between them, thus one might be chosen over the others.

In this thesis work, retransmission’s worst case performances under considered packet loss patterns and various round trip times are compared to performances of forward error correction schemes. In addition, implementation details with respect to the relevant RFCs are provided as an example to give a better judgement on the obtained results.

Results obtained under the packet loss patterns that are generated with a simple Gilbert-Elliot 2-state model show that forward error correction techniques are a reasonable choice of error concealing in the real-time transport protocol (RTP) stack where round trip time in the channel is greater than 200 ms. In addition, bandwidth overhead revealed by forward error correction stays higher than retransmission’s bandwidth overhead in all sample runs. In cases where round trip times are high, then the choice of forward error correction scheme is bound to the packet loss pattern. In the results section, it is obtained that Reed-Solomon performs well in terms of residual packet losses, which are the packets not being recovered, and bandwidth overhead when losses occur in long bursts.