

Evaluation and Implementation of Header Compression Algorithm ECRTTP

Carl Knutsson

Luleå University of Technology
MSc Programmes in Engineering
Department of Computer Science and Electrical Engineering
Division of Computer Communication

Implementation and Evaluation of Header Compression Algorithm ECRTP

Carl Knutsson

November 8, 2004

Abstract

This master's thesis presents an evaluation of *Enhanced Compressed RealTime Protocol* (ECRTP). ECRTP an extension to CRTP, is a header compression scheme for real time traffic. ECRTP exploits the hop-by-hop redundancy in a stream of IP/UDP/RTP packets, where virtually the same header is sent over and over again. The header and additional information is saved in a context at the compressor and decompressor. If the context is synchronized, the compressor can send only the differences since the last header and decompressor will be able to reconstruct the header.

ECRTP was developed to overcome the problems of CRTP. CRTP does not perform well over links with long round trip time that lose and reorder packets. ECRTP extends CRTP by repeating context updates and by sending absolute values along with delta values when encoding monotonically increasing header fields to increase robustness. It inserts a header checksum when UDP checksum is missing, to improve error recovery and fail checks for the compression.

The evaluation of ECRTP consists of two parts, simulations and a theoretical investigation. With the results of the simulations and the theory in mind, weaknesses and strengths are pointed out and suggestions of improvement will be presented.

A possible error in the establishment of the repetition value in the decompressor was found. The error can be avoided by sending the repetition value in the compressed header at the beginning of a session.

The investigation shows that ECRTP handles packet loss and large round trip times well. ECRTP can be configured to handle packet reordering by exclusive use of absolute encoding for monotonically increasing header fields, by using most commonly changing fields that is not protected by the checksum as context-defining fields. The checksum-protected RTP fields should, if changed between two consecutive headers, be included in every packet and not compressed on the assumption of in-order delivery. All the suggested changes will decrease the compression efficiency.

ECRTP is simulated together with two other schemes; CRTP and ROHC. It is shown that *Robust header compression* (ROHC) handles reordering better than expected. Especially the least significant bit encoding shows promising results and the handling of CSRC list. But also the handling of the checksum-protected RTP fields show that ROHC with some modifications could possibly be made to handle compression during reordering more efficiently than ECRTP.

Contents

1	Introduction and Background	4
1.1	Introduction	4
1.2	Theoretical Background	5
1.2.1	Realtime Traffic	5
1.2.2	RTP, Realtime Transport Protocol	5
1.2.3	Header Compression	6
1.2.4	IP/UDP/RTP Change Patterns	6
1.2.5	Classification of the IP/UDP/RTP fields for header compression	6
1.2.6	RTP Flow Identification and Header Compression Context	8
1.2.7	Encapsulating headers	8
1.2.8	CRTP, Compressed Real-Time protocol	8
1.2.9	Packet Loss and TWICE	11
1.2.10	ECRTP	11
2	Discussion, Packet Loss and Packet Reordering	15
2.1	Packet Loss and Reordering	15
2.1.1	Repetition Value Establishment Issue	15
2.1.2	Long Round Trip Times	16
2.1.3	Reordering of Packets	16
2.1.4	ECRTP and Reordering	17
2.1.5	Sequence Number Compression	17
2.1.6	Reordering and Negative TWICE	18
2.1.7	ECRTP and Absolute Encoding (NODELTA)	19
2.1.8	Prelink Reordering and Packet Loss	19
2.1.9	Relative Encoding and State Refresh	20
2.1.10	Least Significant Bits encoding (LSB)	22
2.1.11	Changes in Innermost IP Header and Encapsulating Headers	22
3	Implementation	24
3.1	ECRTP Implementation	24
3.1.1	Packets Formats	24
3.1.2	Determine Packet Type	24
3.1.3	N Repetition in the Decompressor	25
3.1.4	Checksum and TWICE	25
3.1.5	Decompressor Feedback	26
3.1.6	Classifying RTP and Negative Cache	26
3.1.7	Checksums	26

4	Simulations	28
4.1	Simulations	28
4.1.1	Effnet HC-Sim	28
4.1.2	Simulation Setup	29
4.1.3	Sample Traffic	30
4.1.4	Metrics	30
4.2	Simulation Results	31
4.2.1	Audio Stream Simulation	31
4.2.2	Audio Stream Simulation Summary	35
4.2.3	Prelink Reordering	36
4.2.4	Changes in Checksum-Protected Fields	38
4.3	Simulation Summary	39
5	Conclusion	41
5.1	Conclusion	41
5.1.1	The Prerequisites of ECRTP	41
5.1.2	ECRTP Solutions	41
5.2	Future Work	43

Chapter 1

Introduction and Background

1.1 Introduction

This master's thesis covers the implementation and evaluation of a header compression protocol, Enhanced Compressed RTP, ECRTP [1] for Effnet AB. Evaluation consists of two parts, a theoretical investigation and simulations.

ECRTP is a header compression scheme for realtime traffic. Header compression takes advantage of the redundancy between headers in a stream of packets. Instead of sending the whole header over and over again, it sends only differences between consecutive headers or predefined parts of the header that are changing frequently.

CRTP [5] the predecessor of ECRTP, has been shown not to perform well under harsh link conditions. The article, *Evaluation of CRTP Performance over Cellular Links* [2], shows that CRTP does not cope with packet loss very well. Lately, there has been an interest to use header compression not only over single links but also over multi-hop networks. The new demands on the compression algorithm lead to the development of ECRTP. At this date there are two IETF Internet drafts with proposals of how to use header compression over large networks, *Tunneling Multiplexed Compressed RTP* [3] and *Requirements for Header Compression over MPLS* [4].

ECRTP became a standard in July 2003. This new protocol aims to be more robust over links with high round trip times and high frequencies of packet loss. Examples of such links are virtual circuit networks and tunnels. Virtual circuits like ATM, and IP tunnels can have high round trip time and high packet loss rate. These multi-hop links often span over large physical distances, packets can be dropped and reordering can occur in the queues of the switches. ECRTP should be able to handle the large delays, reordering of packets and packet losses in multi-hop networks.

This thesis covers an evaluation of ECRTP. It evaluates if the protocol meets the requirements stated in the RFC3545 [1] and how it performs under different circumstances and environments compared to other header compression schemes. The demands on the compression scheme during link reordering and link packet loss are pointed out. Suggestions to solutions are presented. This survey includes a theoretical study and simulations of ECRTP.

1.2 Theoretical Background

In this section header compression for realtime traffic will be explained. The characteristics of realtime audio and video streams will be pointed out, and the basics of header compression will be explained. ECRTP and its predecessor CRTP will be explained and investigated.

1.2.1 Realtime Traffic

The compression algorithms described in this master's thesis, are all compressing realtime traffic. Realtime traffic, such as audio and video, is associated with some problems. Realtime packets have to be delivered within a time bound and are also sensitive to jitter and packet loss.

The payloads of audio and video packet are generally quite small. In audio case, the small payload is to reduce the delay. Only a few sound or video samples are sent in each packet. If the sound or video is delayed too much the impression of realtime communication is lost, and interaction between two parties will be difficult.

Traditionally, problems with packet loss is solved by receiver triggered resends and buffering, i.e. TCP. In realtime traffic, such a mechanism would only introduce more delay and when the resent data arrives it will be obsolete and to no use for the receiver. Buffering in the receiving application can be used to solve some of the problems caused by reordering. However, large buffers can introduce large delays and the receiving application usually drops packet arriving too late in a realtime stream.

To be able resolve the problems associated with a realtime transport, the protocol should have some mechanism to determine the order of the packets, so that the reconstruction of the realtime stream is possible. With sequence number, the receiver can determine the order of the packets. With a timestamp, the jitter can be remedied, and the receiver can recreate the realtime stream.

Realtime traffic can be encoded in different ways. Different codec encodes the samples differently. The number of samples or frames in each packet varies. Sometimes redundancy is included and in some codec the payload is compressed.

1.2.2 RTP, Realtime Transport Protocol

RTP[6] is a realtime transport protocol. It can be used to send audio and video stream over unicast or multicast channels. RTP is meant to be a generic realtime transport protocol for realtime traffic, such as video, audio and simulation data. RTP can carry different payloads and supports many different codec. It also has functionality to multiplex several RTP streams into one and demultiplex it back to multiple streams.

RTP has two types of packets, RTCP control packets and RTP data packets. RTP Control Protocol (RTCP) primary function is to provide feedback on the quality of the data distribution. It can be used to adapt to the network environment and to diagnose faults in the distribution. It also distributes information about participants in an RTP session. RTCP headers are not compressed in any of the header compression schemes and will not be further discussed in this master's thesis. The RTP data packets only function is to carry the audio/video samples or frames. The remainder of this master thesis will discuss different strategies and issues of compressing the header of the RTP data packet.

1.2.3 Header Compression

Header compression takes advantage of the hop-by-hop redundancy of the headers in a flow of packets over a link. In an IP/UDP/RTP packet stream, the same fields are sent over and over again, even if most of the fields in the headers are unchanged. Some fields are used by the network and are of no use when sent over one hop. Instead of sending the whole header, a compressed header can be sent. The compressed header contains information about changes in the header since the last packet. The receiver can then reconstruct the packet using the previous header and pass it on to the network. The previous header and other data, used to compress or decompress, are saved in a context at the sender and the receiver. A compression scheme like this is depending on that all packets are delivered successfully to the decompressor without reordering.

Most header compression schemes have several packet types. Some of them must have support from underlying layers to communicate the packet type of a compressed packet. If not supported by the link layer, PPP[7] can be used to distribute the packet type to the receiver. CRTP and ECRTP are dependent in this way on the link layer.

Before header compression can be started on a link, a set of configuration parameters has to be negotiated between the participants of the compression. RFC3241[8] and RFC3544[9] specify how this is done over PPP.

1.2.4 IP/UDP/RTP Change Patterns

As explained above, many of the fields in the RTP header and in encapsulating headers are constant or change in a predictable manner. Most of the fields are actually not changing at all or changes very rarely. In the IP and UDP headers, all fields are normally constant, checksum and length fields excluded, throughout an RTP session. The fields that are normally not changing over a link are, the Source and Destination Port for UDP and the Version field, the Protocol field, Time To Live field, Type of Service field for IPv4. For IPv6, the fields are; the Version, Traffic Class, Flow Label, Hop Limit and Next Header.

Some of the fields in an IP/UDP/RTP header are dependent on other fields or on the payload of the datagram. For IPv4 these dependent fields are, the Header Length, the Total Length and the Header Checksum and for IPv6 the Payload Length. UDP includes two fields that have this dependency, Length and Checksum. The RTP header has no fields that are dependent of other fields in this way, except for perhaps Contributing Source Count.

1.2.5 Classification of the IP/UDP/RTP fields for header compression

In header compression, the fields are classified into groups, depending on their changing pattern between headers in flow. The fields in CRTP and ECRTP can be grouped into four classes: NOCHANGE, DELTA, RANDOM and INFERRED.

The IPv4/UDP/RTP header fields are grouped by CRTP as described in the figure 1.1. The majority of the fields in the header are not changing between packets in IP/UDP/RTP stream. That goes for both IPv4 and IPv6. The exceptions are the IPv4 Identification, the RTP Timestamp and the RTP Sequence Number that changes in virtually every packet. This gives an indication that some of this field can be left out and not sent in every packet.

NOCHANGE	Fields that are not expected to change between headers of the same flow.
DELTA	Fields that changes often with small, often predictable values.
RANDOM	Fields that are changing randomly, in a manner that is unpredictable.
INFERRED	Fields that are inferred from other values in the header or payload.

Figure 1.1: Example: IPv4/UDP/RTP Header classified for CRTP

Version	IHL	TOS	Total Length	
Identification			Flags	Fragment Offset
Time to Live	Protocol	Header Checksum		
Source Address				
Destination Address				
Source port		Destination Port		
Length		Checksum		
Ver	P	X	CC	M
Payload Type			Sequence Number	
Timestamp				
Synchronization Source (SSRC) Identifier				
Contributing Source (CSRC) Identifiers				
...				

= NOCHANGE
 = INFERRED
 = DELTA

In figure 1.1 the fields that are classified as INFERRED are IPv4 fields Total Length and Header Checksum, and the UDP fields Length and the UDP Checksum. The fields that are classified as DELTA are the IPv4 Identification and RTP Sequence Number and RTP Timestamp. The rest of the fields are considered to be NOCHANGE.

In IPv4, the Identification field will be implemented differently in different IP stacks. It is supposed to be a unique identifier for every datagram. One solution is to increase the IPv4 Identification by one. It can be increased either by “per-stack” basis or “per stream” basis. It can also be byte swapped sequentially for each IPv4 packet created. The IPv4 Identification is used for fragmentation, since IPv6 does not support fragmentation it has no such field. In most systems, for example in the BSD¹ stack, the IPv4 Identification is increased by one and can be classified as a DELTA field. It can also be RANDOM and in some specification violating Linux stacks it has been shown to be NOCHANGE

Initially, the compressor sends one or more packets containing a full header and additional information to set up the decompressor. When the initialization is done, the compressor can start sending compressed packets. The compressed packets only contain the fields that have changed since the previous packet. The decompressor can reconstruct the compressed header using the previous header. If a NOCHANGE field has changed since the last packet, it is communicated to the decompressor, by sending

¹Berkeley Software Distribution (4.4BSD) operating system (Unix-variant).

the packet with a complete IP/UDP/RTP header again.

The UDP checksum is often sent “as-is” in a compressed packet because of the end-to-end reasons. If not sent, the decompressor could unintentionally reconstruct an erroneous packet with a seemingly correct checksum and thereby violating the end-to-end argument.

If a packet is lost and the decompressor is unable to reconstruct next packet, all reconstruction of subsequent packets is likely to fail. Therefore, the decompressor needs to be able to send feedback to the compressor. The compressor will then react on the feedback and send a complete header to the decompressor.

1.2.6 RTP Flow Identification and Header Compression Context

Header compression is usually point-to-point, used over one link. There is a possibility of several IP/UDP/RTP flows over the same link. The compressor and decompressor need to keep a context for each compressed flow. The header of the previous packet and other data needed for the compression or decompression are saved in the context. A flow can be identified by the NOCHANGE fields. The fields that identifies if a packet belongs to a context is called context-defining fields. Each flow is given a unique identifier, context id (CID). The CID is sent in every compressed packet to enable the decompressor to determine which flow the packet belongs to. The context for CRTP and ECRTP contains a link sequence number and a generation number. The sequence number is incremented for every packet and the generation number is incremented when a new flow is started for a context.

1.2.7 Encapsulating headers

A packet arriving to a compressor can have additional encapsulating headers, for example routing headers or authentication headers. These headers can be compressed in CRTP and ECRTP and are classified in RFC2507[10]. Encapsulating IP header are compressed differently from innermost IP header. The non-static fields or the encapsulated headers are sent as randomfields.

1.2.8 CRTP, Compressed Real-Time protocol

CRTP is designed to compress IP/UDP/RTP flows. CRTP uses four packets formats: Full Header, Compressed UDP, Compressed RTP and Context State. In this section CRTP will be briefly described. A more detailed description can be found in RFC2508[5].

Full Header

The Full Header is used to establish a flow context in the decompressor. It is used by the compressor at the beginning of a flow and when the decompressor request feedback to reestablish synchronization of the contexts. The Full Header packet is normal IP/UDP/RTP packet with a few changes. The first two length fields are used to send information about the flow to the decompressor. The information sent is the connection identifier, a generation number and a sequence number. The connection identifier (CID) can be 8 or 16 bits.

The first two length field are usually an IP length field and a UDP length field, when there are no encapsulating headers.

Figure 1.2: First and second length field for 8-bit context ID



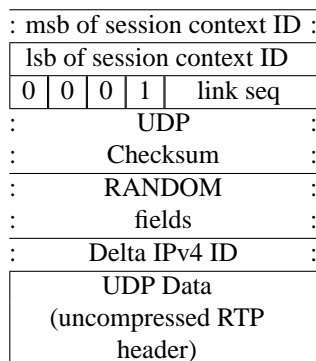
Figure 1.3: First and second length field for 8-bit context ID



Compressed UDP

Compressed UDP is used when a normally constant NOCHANGE value in the RTP header changes. The RTP Payload Type is an example of a NOCHANGE field. The IP header, the UDP header and encapsulating headers can be compressed while the RTP header is sent as is in the packet. It can also be used to compress IP/UDP that does not carry RTP.

Figure 1.4: First and second length field for 8-bit context ID



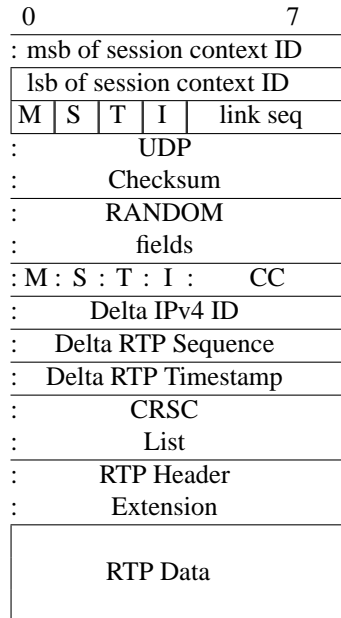
The header contains a CID for the flow, a bitmap to indicate first order changes in the IPv4 Identification field and a link sequence number. The UDP checksum is inserted to allow validation of the header reconstruction in the decompressor. Random fields are always included for all encapsulating headers. The new delta IPv4 Identification is included only if I bit is set. For IPv6, the I-bit is always set to zero. A full RTP header and payload should be inserted after the compressed IP/UDP header.

Compressed RTP

The Compressed RTP header includes the whole IP/UDP/RTP. A compressed RTP header can be sent when all NOCHANGE values except CSRC count and CSRC-list are constant. It can handle changes DELTA encoded fields and it also handles changes in the marker bit.

The Compressed RTP header is similar to the Compressed UDP. Compressed RTP has more flags in the bitmap that follows the CID. The bitmap includes M,I,S,T flags for RTP Marker bit and first order difference of; the IPv4 Identification, the RTP Sequence Number and the RTP Timestamp. The fields for IPv4 Identification, RTP Sequence number and RTP Timestamp follow the UDP Checksum and Randomfields in the header. If all flags are set in the bitmap that indicates a change in the CSRC list,

Figure 1.5: CRTP Compressed RTP header

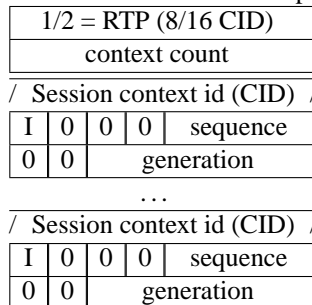


and a second bitmap is included that contain the real values of the M,I,S,T flags and the CSRC count. The new CRSC list is inserted after the second bitmap in the header. If RTP Extensions is used it is included last in the header and is followed by the payload.

Context State

Context State packets are sent by a decompressor as feedback to the compressor. It is used to notify the compressor that the decompressor suspects that it has an invalid context. An entry is included for each of the involved flows. The update request can either be advisory or mandatory. The Context State packet is also called header request.

Figure 1.6: CRTP Context state packet.



The first byte in the header request indicates if the CID is 8 bits or 16 bits. The next byte contains the number of context entries included in this Context State. A context

entry consists of a CID, an I-bit, the link sequence number and a generation number of the context. If the I-bit is zero, the Context State is advisory. Header request is another name for the context state that will be used throughout this report.

1.2.9 Packet Loss and TWICE

As mentioned above, the loss of the wrong packet can cause reconstruction failures in the decompressor. The context synchronization is lost when a packet containing a context update is lost. The most common changes between headers in an IP/UDP/RTP flow, is the IPv4 Identification field, RTP Sequence Number field, RTP Timestamp field and the RTP Marker Bit. The marker bit is encoded in every packet in CRTP and is not saved in the context. A change in the marker bit cannot force the context in the compressor and decompressor out of sync.

IPv4 Identification, RTP Sequence Number field and RTP Timestamp changes in almost every packet. These changes are often predictable, the RTP Sequence Number should be incremented between every packet and the RTP Timestamp is often monotonically updated. With this knowledge, a lost packet's header can be compensated with large probability. The delta values in the decompressor context can be added to the saved header to make up for the lost packet. The UDP checksum of the packet can be used to validate the guess. This algorithm is called TWICE. IPv4 Identification is not covered by the UDP checksum and caution must be exercised when using TWICE. A more detailed description of the TWICE algorithm is given in RFC2507[10].

It should be mentioned that the TWICE-algorithm can only manage the loss of packets containing predictable changes in the DELTA fields. It cannot handle loss of changes in NOCHANGE fields.

1.2.10 ECRTP

There are three significant changes in RFC3545 compared to RFC2508. First, all updates for DELTA values are sent as both delta and absolute values. Second, a new checksum, HDRCHKSUM, is inserted into all compressed packets when the UDP checksum is missing. Third, all updates in the context are replicated in several compressed headers to secure the establishment of the updates.

Updates to Delta Encoded Fields

The format of the Compressed UDP packet is changed to be able to encode DELTA values as both the difference since the last value and the absolute value. In ECRTP, all changes in DELTA fields are encoded as either an absolute value or as both a delta value and an absolute value. When a Compressed UDP packet is received by the decompressor, the decompressor will be able to reconstruct a correct context for the delta value even if earlier updates of that delta value were lost. This is done to decrease the number of context invalidations due to packet loss and reception of out of order, late packets.

Header Checksum (HDRCHKSUM)

A UDP packet can be sent with a zero checksum. That will make validation of TWICE recovery in the decompressor impossible. To cope with a missing UDP checksum, the compressor inserts a new header checksum into every compressed packet. The

header checksum is a 16-bits one-complement checksum, calculated over the pseudo IP header, as specified in RFC768[11] and the UDP header plus the RTP header including the CSRC list. The payload is not included in the checksum. If the packet is an IP/UDP packet and the RTP header is missing, 12 bytes or as many as are available of the payload is used instead.

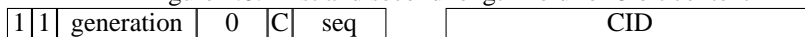
Full Header

The Full Header packet only includes some small changes compared to CRTP. The first two-length field encountered when parsing the encapsulating headers that contain the context information, are subject to a change in format. The format has changed to give room for the C-flag. The C-flag indicates the use of the Header Checksum described above. The new format of the length fields is shown in figure 1.7 and figure 1.8.

Figure 1.7: First and second length field for 8-bit context ID



Figure 1.8: First and second length field for 8-bit context ID



Compressed UDP

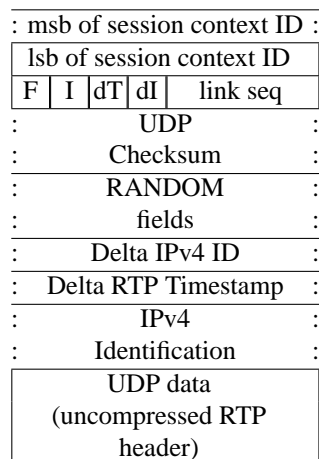
ECRTP uses two types of Compressed UDP, Traditional and Extended. The Traditional Compressed UDP compresses only IP/UDP fields and can update all first order differences to the DELTA fields in the context. The Extended Compressed UDP compresses the whole IP/UDP/RTP header and can update all first order differences to the DELTA fields.

The traditional Compressed UDP can be used to update NOCHANGE fields in the RTP header and refresh all DELTA values. It can also be used to compress IP/UDP packets without an RTP header.

The ECRTP Traditional Compressed UDP has a change in the bitmap compared to Compressed UDP in CRTP. The Traditional Compressed UDP compresses the IP/UDP headers and leaves the RTP header untouched. The bitmap is changed and includes the flags: F,I,dI,dT. The F-flag is zero for Traditional Compressed UDP and one for Extended Compressed UDP. The I-flag has changed compared to CRTP. It indicates that the absolute value of IPv4 Identification is included in the packet. A new flag dI is used for the first order difference of the IPv4 Identification. The dT-flag indicates that the first order difference of RTP Timestamp is included. The first order difference of RTP Sequence Number is expected to always be one and is not included in the packet.

The Extended Compressed UDP is indicated by the F-flag and differs from the Traditional compressed UDP by compressing the actual RTP header. It is sent as a Compressed UDP for link layer purposes, but is actually more like Compressed RTP packet, since it compresses the whole IP/UDP/RTP header. It includes an additional bitmap that indicate inclusion of some of the RTP header fields. The bitmap includes; the RTP Marker, a S-flag for absolute RTP Sequence Number, a T-flag for the absolute RTP Timestamp, a P-flag for RTP Payload Type and a C-flag for inclusion of an additional byte containing the CSRC count.

Figure 1.9: ECRTP Compressed UDP header “Traditional”(F=0).



Both the Traditional and the Extended Compressed UDP can be used to send both first order difference and absolute value for the IPv4 Identification and the RTP Timestamp. To be strict, the Extended Compressed UDP packet is actually a Compressed RTP, because it compresses the RTP header, but it extends the Compressed RTP by being able to send the RTP Payload Type and absolute values for the DELTA fields.

Compressed RTP and Context State

The Compressed RTP packet format is the same in ECRTP as in CRTP (see section 1.2.8). Compressed RTP is only sent when there are no changes to the context's delta fields. It is allowed to send changes to marker bit and changes in CRSC-list.

The context state packets have the same format as in CRTP.

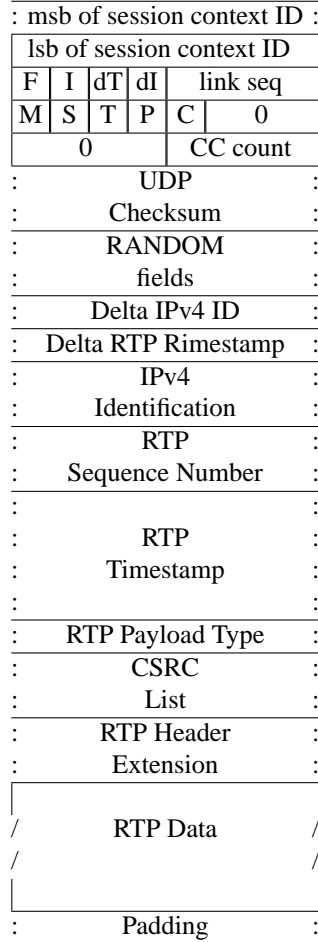
Robust Operation, N+1 sequence

To be able to handle packet loss during the establishment of a new context or during updates to the context, the ECRTP compressor sends every update to the context in several subsequent packets.

A number “N” is specified to represent the quality of the link. It means the probability of losing N adjacent packets on the link is very small. So if every update to the context is included in N+1 consecutive packets, the decompressor will, with a large probability keep its context synchronized. Losses less than N can be resolved by using the TWICE algorithm.

ECRTP starts a new flow by sending the first N+1 packets as Full Header to initialize the decompressor context. It is followed by N+1 Compressed UDP packet to establish the delta values in the context. The decompressor learns the value of N by counting the number of Full Header. If an unpredictable change occurs during the Full Header repetition, it is immediately interrupted and a new N+1 Full Header send sequence with a new generation number is started. This is to avoid getting an incorrect N value in the decompressor.

Figure 1.10: ECRTP Compressed UDP header “Extended” (F=1).



The decompressor needs the N for two reasons. First, when TWICE is used more than N times, the IPv4 Identification cannot be guaranteed, since it is not covered by the UDP checksum or the HDR checksum. So, for IPv4 it is not wise to TWICE more than N times. Second, the decompressor uses N when the context is lost and it needs a new context from the compressor. N+1 context state packets are sent back to compressor to trigger a Full Header update sequence. N+1 context state is sent to assure that at least one context state arrives at the compressor.

Chapter 2

Discussion, Packet Loss and Packet Reordering

2.1 Packet Loss and Reordering

This section will discuss the characteristics of packet reordering and packet loss and their impact on the header compression algorithms. Weaknesses and strengths will be discussed and solutions or parts of solutions will be suggested. Existing solutions within the ECRTP framework will be presented as well as a solution that would require changes in RFC3545.

2.1.1 Repetition Value Establishment Issue

In section 1.2.10 of previous chapter, the establishment of the repetition value N is described. As the observant reader may have discovered, the repetition value establishment can fail even if less than N compressed packet are lost on the link. If the last packets of the repetition value establishment sequence is lost the N will be smaller in the decompressor than in the compressor. Suppose that the decompressor discovers that the context is lost and start sending $N+1$ header request back to the compressor. Since the decompressor has a repetition value that is smaller than N , the whole header request can be lost even if less than N packets is lost. Reordering can temporarily cause the decompressor to use an incorrect repetition value as well. The RFC acknowledges the problem by claiming incorrectly that the only drawback of a smaller N in the decompressor will be that it will start sending header requests if fewer packets than N are lost. The N establishment of counting Full Header is not a sufficient solution.

A possible solution could be to find a lower bound for N by counting Full Header and checking the generation number. An upper bound for N can be found by observing the link sequence number in the first arriving non Full Header packet that is successfully reconstructed. The lower bound is used when determining if a header request transmission sequence should be started. The upper bound is used when sending the header request repetition sequence.

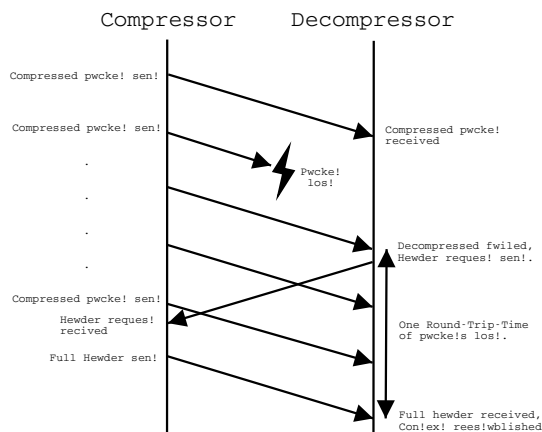
Another solution to this problem is to send N in the length field of the Full Header. For 8-bits CID there are 11 bits that are unused and for 16-bits CID there are three unused bits. So, we have three bits to encode the N . This gives the interval of between 0 and 7. This should be sufficient. However, this solution limits the choice of N , but

instead the N criterion hold.

2.1.2 Long Round Trip Times

Long delays or long round trip times is a problem for most of the header compression algorithms where the compressor is dependent on feedback from the decompressor. In CRTP and ECRTP, when the contexts in the compressor and decompressor are put out of sync, the decompressor sends a context state packet back to the compressor. The context state packet contains the flows that are out of sync. The compressor can then send a Full Header to reestablish the contexts of the invalid flows. A whole round trip time of packets is lost. Half a round trip time for the context state packets to travel from the decompressor to the compressor and another half for the Full Header to reach the decompressor again.

Figure 2.1: Packet loss between compressor and decompressor in ECRTP

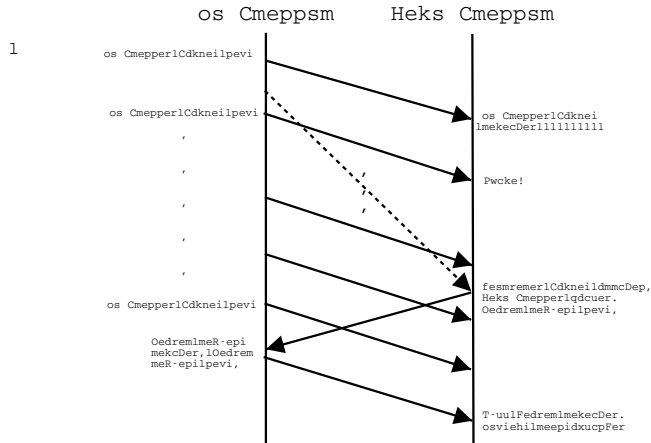


2.1.3 Reordering of Packets

Reordering occurs when a packet is delayed so that one or several subsequent packets arrive at the destination before the delayed Packet. When a packet is delivered out of order to the decompressor, the same scenario occurs as when a packet is lost.

If the delayed packets include only predictable updates to the context, the TWICE algorithm can be used to recover from the missing packet. The context is saved, but when the out of order packet arrives to the decompressor it will fail the reconstruction of the header. The compressor will then try to reestablish the context by sending a context state. The contexts are actually in sync and the consequence will not be as severe, because subsequent packets can be reconstructed. However, the compression rate will decrease if the decompressor has to send context state packets for every reordered packet. The compressor has to send at least N+1 of the more redundant Full Header on the reception of a context state packet instead of a Compressed UDP or Compressed RTP.

Figure 2.2: Reordering between compressor and decompressor in ECRTP



2.1.4 ECRTP and Reordering

The previous section 2.1.3 gives an explanation of what happens when a packet is re-ordered. First there will be a sequence number gap that in many cases will be solved by the TWICE algorithm. If an update to the decompressor is delayed, the decompressor will fail to reconstruct any arriving packets until the context is reestablished. ECRTP solves this problem by allowing the decompressor to validate packets with TWICE after a decompression failure. When the out of order late packet arrives at the decompressor, the decompression will fail as described in the previous section. The ECRTP decompressor is allowed to try to decompress subsequent packets and the context can be saved if no updates are lost.

It is not always possible for the decompressor to determine if a packet is arriving late, out of order, or if the context updates are lost and the context is invalid. The ECRTP decompressor must always send a Header Request (Context State) back to the compressor to ensure that it would not fail to reconstruct more packets than necessary.

However, this solution is not waterproof. As mentioned above ECRTP does not handle a situation where an update is reordered. An ECRTP decompressor does not only lose the reordered packet, it sends a context state for every reordered packet, which will result in a less efficient compression. It is an improment compared to CRTP. CRTP loses every packet after a failed reconstruction until it receives a complete context update from the compressor.

2.1.5 Sequence Number Compression

ECRTP has a four bits link sequence number inserted in every compressed packet for a context. The small sequence number space increases the risk of wraparound. As mention in the previous section, it cannot be determined if an arriving compressed packet is reordered or not. TWICE can sometimes be used to resolve if the packet is delivered in order or not, but if packets have been lost it can be impossible.

A solution to this problem is to increase the sequence number space in the compressed header. A sequence number of 8 bits is probably sufficient for most applications. In RFC2507[10] section 10 a solution is suggested, where an 8 bits sequence

number is inserted in every compressed packet.

ROHC[12] has a more efficient compression technique that could be applied to handle reordering. It uses the RTP Sequence Number, instead of the link sequence number, when compressing the header. It is easier to find a relation between RTP Sequence Number and the IPv4 Identification or RTP Timestamp than between a link sequence number and IPv4 Identification or RTP Timestamp. If prelink reordering occurs on an RTP audio stream, it is unlikely to be a relationship between the link sequence number and the RTP Timestamp, while a relationship between RTP Sequence Number and RTP Timestamp is very likely. As a result, it is sufficient for ROHC to send only the RTP Sequence Number in many situations where ECRTP would have to send both IPv4 Identification and RTP Timestamp along with the RTP Sequence Number.

2.1.6 Reordering and Negative TWICE

The TWICE algorithm is normally used by the decompressor to recover from packet loss. However, TWICE could be used to validate reordered packets too. When an out of order, late packet arrives to the decompressor, the DELTA encoded fields have already changed in the context. Successive packets have updated the fields making it impossible to use TWICE. This will cause the decompressor to send a context state and the compression efficiency will decrease. One possible solution could be to use negative TWICE. To subtract the first order difference to the DELTA fields similar, opposite to TWICE. This does not solve the situations where a normally constant field, such as the Payload Type, has changed when the out of ordered packet arrives to the decompressor.

Figure 2.3: ECRTP Negative TWICE, Packets arriving to decompressor

Pkt	seq	I	T	S	dI	dT	Decompressor Context
CU	1	1	0	2	1	1	I=1, T=0, S=2
CU	2	2	1	3	1	1	I=2, T=1, S=3
CR	3	-	-	-	-	-	I=2+1=3, T=1+1=2, S=3+1=4 Apply delta
CR	4	-	-	-	-	-	I=3+1=4, T=2+1=3, S=4+1=5 Apply delta
CR	6	-	-	-	-	-	I=4+2=6, T=3+2=5, S=5+2=7 TWICE
CR	7	-	-	-	-	-	I=6+1=7, T=5+1=6, S=7+1=8 Apply delta
CR	5	-	-	-	-	-	I=7-2=5, T=6-2=5, S=8-2=6 Negative TWICE

The scenario in Figure 2.3 was saved by the Negative TWICE scheme. When the reordered packet, with link sequence 5, arrives to the decompressor the first order difference was multiplied by the link sequence number and added to the existing absolute values. In this case the sequence number delta was negative. Negative TWICE for IPv4 is not encumbered by the N+1 limit. It is limited by last change in a constant field or a change in first order difference for a delta field.

When recovering with negative TWICE, the packet should not update the context. Field that are not protected by checksum can be changed by the late packet. Such change in the decompressor's context could cause the decompressor to reconstruct all successive packets incorrectly. This can be avoided if the number of packets since last change to context is saved. However, if the sequence number is updated in the decompressor's context, then IPv4 packet could be incorrectly discarded by the N+1 TWICE limit, leading to an unnecessary context invalidation. A packet recovered by Negative TWICE, could if allowed include an old, outdated change into context. Hence, if using Negative TWICE, packets should not update the context of the decompressor.

Both TWICE and Negative TWICE can of course only be used to recover the delta fields. Negative TWICE is also dependent on that the delta field is changing monotonically. If reordering before the compressor occur, new first order differences have to be sent more often by the compressor, thus limiting the possibility to use TWICE or Negative TWICE in the decompressor.

2.1.7 ECRTP and Absolute Encoding (NODELTA)

ECRTP can be configured to handle some of the problems associated with reordering. If configured to only send absolute values, context invalidation caused by the fluctuations in the normally monotonically increasing DELTA fields, can be avoided. These irregularities in the header flow can be caused by several reasons. Two causes could be reordering or silence suppression. If the compressor always sends Compressed UDP packets with the F,I,S,T flags set for Extended or I flag set for Traditional, the decompressor can reconstruct the IPv4 Identification, the RTP Sequence Number and the RTP Timestamp for the scenario described in the next section. This way, compression efficiency will be traded for robustness against reordering. This configuration will be denoted as ECRTP NODELTA.

The configuration will not resolve all problems of reordering. Changes in other field than the DELTA fields will be encoded the same ways as ECRTP in “normal” mode. If a full sequence of Full Header or updates to the Payload Type or the CSRC list is lost or delayed, ECRTP NODELTA will still fail or reconstruct erroneous packets.

2.1.8 Prelink Reordering and Packet Loss

As discussed earlier, the most common changes between two successive packets in RTP are the delta values. The header compression schemes CRTP and ECRTP establish first order difference in the context to be able to compress more efficiently. The TWICE algorithm also take advantage of this property, when trying to reconstruct compressed packets. It will only succeed if the first order delta is unchanged.

When reordering and packet loss occur before the compressor, the DELTA fields will not monotonically increase in a predictable manner. This has the effect that the compressor has to send first order difference of the DELTA fields more often than for a flow that is not reordered. The compression efficiency will decrease and the compression will be more vulnerable to reordering and packet loss between the compressor and the decompressor, since the main functionality to handle these problems is TWICE. However the compressor will start to send more redundant Compressed UDP packets that include absolute encoded DELTA fields. It is unclear how this behavior affects the robustness. Larger packets suggest increasing robustness, but sending more updates to the context increases the number of situations where decompression can fail.

The same thing will happen in an IPv4/UDP/RTP stream coming from a stack that increases IPv4 Identification per stack and not per stream. If the stack send an IP packet not belonging to the stream, the stream will have an IPv4 Identification difference that are not constant between successive packets. An unpredictable change in the IPv4 Identification field will not be discovered by TWICE, since it is not covered by the UDP checksum. The decompressor will then reconstruct an erroneous header instead. The incorrect IPv4 Identification will be stored in the header in the context, causing the decompressor to reconstruct incorrect headers until the IPv4 Identification is updated again.

To conclude, reordering of packets before the compressor could make the compression more vulnerable to harsh link conditions between the compressor and the decompressor due to the first order difference encoding. A compression scheme that is robust to reordering and packet loss has to use another scheme to compress the DELTA fields. The compression efficiency will decrease if prelink reordering occurs. It is not obvious how prelink reordering affects the robustness of ECRTP compression scheme.

2.1.9 Relative Encoding and State Refresh

Another way to tackle the problems with reordering is to send the absolute values of the DELTA encoded fields in the header and send compressed values for the DELTA fields encoded, not as the difference since the last packets, but instead it is encoded as the difference since the last absolute value sent.

This would require some changes in the protocol of ECRTP. N+1 Full header will be sent at start up, in exactly the same way as in ECRTP. The N+1 Compressed UDP that follows the Full Header sequence will use the absolute value of the first of the Compressed UDP packets as absolute value reference. The DELTA fields will then be encoded as the difference from that value. When N+1 Compressed UDP packets are sent, the absolute value of each DELTA field is saved in the context. Now subsequent packets can encode the DELTA values as the difference from the established absolute value. The difference will be encoded the same way as described in RFC3545. The UDP checksum or the HDR checksum must always be present in the compressed packets to be able to validate the reconstruction.

Figure 2.4: ECRTP relative encoding example

Pkt	I	T	S	dI	dT	dS	Context
FH	-	-	-	-	-	-	Set constant values
FH	-	-	-	-	-	-	
FH	-	-	-	-	-	-	
CU	0x4	0x6	0x2	-	-	-	I=0x4, T=0x6, S=0x2
CU	0x4	0x6	0x2	0x1	0x4	0x1	I=0x4+0x1, T=0x6+0x4, S=0x2+0x1
CU	0x4	0x6	0x2	0x2	0x8	0x2	I=0x4+0x2, T=0x6+0x8, S=0x2+0x2
CR	-	-	-	0x3	0x12	0x3	I=0x4+0x3, T=0x6+0x12, S=0x2+0x3
CR	-	-	-	0x4	0x16	0x4	I=0x4+0x4, T=0x6+0x16, S=0x2+0x4
CR	-	-	-	0x5	0x22	0x5	I=0x4+0x5, T=0x6+0x22, S=0x2+0x5

The figure 2.4 shows an example of how ECRTP would work with relative encoding. The figure shows the packets sent from the compressor to the decompressor and how the packets are interpreted. The repetition value N is 2. The Full Header N+1 sequence sets up the connection in exactly the same way as ECRTP. The absolute values of the DELTA encoded fields IPv4 Identification, RTP Sequence Number and RTP Timestamp are established as 0x4, 0x6 and 0x2 in N+1 compressed UDP packets. The context is established at the decompressor and the compressor can now start sending Compressed RTP packets containing the difference from the established absolute values.

The advantage with this method against a NODELTA encoding is that the delta can be encoded in variable length. The relative encoded has a significantly smaller minimal packet than ECRTP NODELTA. The difference in size can be studied in figure 2.1.9

and 2.1.9.

ECRTP NODELTA uses a Extended Compressed UDP containing, a 1 octet CID, a 2 octets bitmaps and link sequence number, a 2 octets checksum, 2 octet to encode IPv4 Identification field, 2 octets for the RTP Sequence Number and 4 octets to encode the RTP Timestamp. The scheme, with relative encoding described in this section can be compressed in a Compressed RTP. It contains, 1 octet CID, 1 octet bitmap, 2 octet checksum, 1 octet IPv4 Identification, 1 octet RTP Sequence Number and 1 octet RTP Timestamp.

Figure 2.5: ECRTP NO DELTA Compressed UDP header (smallest packet)

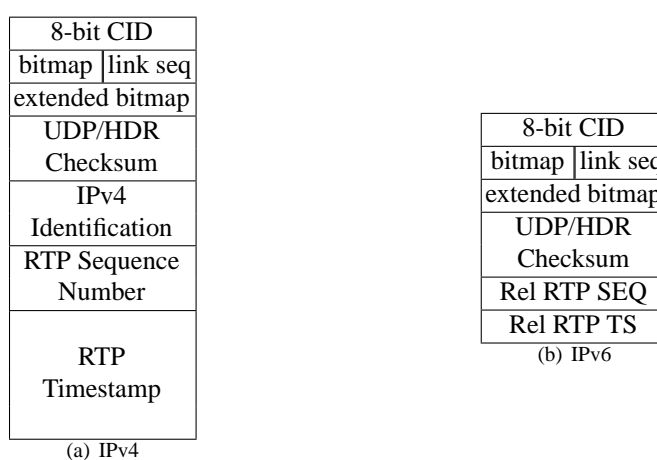
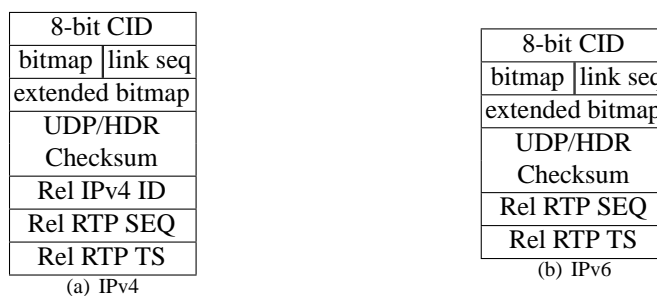


Figure 2.6: RTP relative encoding



The minimal packet for ECRTP NODELTA is 13 octets for IPv4 and 11 octets for IPv6. The minimal packet for relative encoding is 7 octets for IPv4 and 6 octets for IPv6. The relative scheme compressor cannot send this minimal packet as often as the NODELTA scheme. The difference to the established absolute value will grow larger for each packet and the compression efficiency for the relative encoding will eventually be the same as NODELTA. This can be solved by refreshing the absolute in

the context and establish a new by sending N+1 sequence of Compressed UDP when the decompression rate decreases.

The relative encoded header compressor algorithm is more vulnerable to reordering between the compressor and decompressor than the FIST scheme. If a compressed packet that should have arrived before an establishment of a new absolute value, arrives after the establishment it cannot be reconstructed. The decompressor will then send a context state to reestablish the whole context. This can be solved by saving earlier established absolute values and try to recover using those.

2.1.10 Least Significant Bits encoding (LSB)

Least Significant Bits (LSB) encoding could also be successfully used for handling the delta values during link reordering. Below follows a short description of LSB encoding. A more detailed description can be found in RFC3095[12].

Least significant bits encoding, encodes only the k least significant of the value. The decoder can then derive the value using an interpretation interval and a reference value. The size of the interval is 2^k and the reference is previously negotiated value. The reference value could be the last decoded value. The interval can be defined as below.

$$f(v_{ref}, k) = [v_{ref} - p, v_{ref} + (2^k - 1) - p]$$

Figure 2.7: Example of LSB encoding, $k = 2$, $v_{ref} = 0xaf01$, $p = 1$

$k = 2$, $v_{ref} = 11010101$, $p = 1$ gives the interpretation interval: (11010100, 11010101, 11010110, 11010111)

The binary value 11010111 is encoded 11 and gives a new interval: (11010110, 11010111, 11011000, 11011001)

In ROHC the p -value is rather small to be able handle large increases of the value. These large increases can be caused by packet loss or by silence suppression schemes. When silence suppression is used the RTP Timestamp can differ significantly between two consecutive packets. However, to be able to handle reordering, p should be larger so that the interval is more centered around the reference value. An interpretation interval more centered around reference will allow more late reordered packet to be decoded. Late packets can be within interval can often be saved by the decompressor.

2.1.11 Changes in Innermost IP Header and Encapsulating Headers

When changes in fields that are not protected in checksum occur, the compressor must send a full repetition sequence of N+1 Full Headers to establish the new value. As discussed earlier, this is not only inefficient it is also a problem when packets are re-ordered on the link between the compressor and the decompressor. Packets can be incorrectly reconstructed if a packet, sent before a change, arrives at the decompressor after the change is realized. Some of these fields have a tendency to change back and

forth between the same values. It is important for efficiency reasons and to minimize reconstructing incorrect packets. To frequently introducing new values to fields that is not protected by the checksum in the decompressor context increases the number of situations where packet loss and packet reordering can put the contexts out of sync.

There is a possibility that the Type of Service field (TOS) and Time to Live (TTL) field changes in the innermost IPv4 during lifetime of an RTP session. The IPv6 field; Traffic class, Hop Limit and Flow label could also change during an RTP session. Changes in these fields are unlikely for IP/UDP/RTP. These fields are not protected by the UDP or header checksum and must be compressed with care. Guidelines on how to handle these fields over links with packet loss and reordering will be presented in this section.

The IPv6 Hop Limit and the IPv4 TTL is decremented for every node it passes on the way to its destination. Both these fields can during a RTP session, be changed back and forth due to for example route flaps. Therefore it is recommended to use the IPv4 TTL or IPv6 Hop Limit as a context-defining field. The RTP stream will be split into several contexts instead of sending Full Header every time TTL or Hop Limit changes.

The IPv4 TOS field and IPv6 Traffic Class are used for differentiated services and are not expected to change. The TOS and Traffic Class field can change and should be used as context-defining fields to avoid the problems described above. However, the IPv4 TOS is also used for explicit congestion notification. Currently this is not used for UDP, but it could be wise to avoid using these two bits for as a context-defining fields. The IPv6 Flow Label is an identifier for the IPv6 flow. So, the IPv6 flow identifier is a good candidate for a context-defining field. IPv4 options and fragmentation is not taken into consideration, since packet with IPv4 options or IPv4 fragment will be sent uncompressed.

The chain of encapsulating headers should also be used as a context-defining field. All the fields in the headers should be absolute encoded. Changes in encapsulating header should be considered as rare, and several contexts can be allocated if the chain of encapsulated headers is changed.

Chapter 3

Implementation

3.1 ECRTP Implementation

The ECRTP implementation discussed in this section is built upon Effnet's CRTP source code. The design will not be described in detail, but some issues that came up during the implementation will be discussed. As described in section 1.2.10, there are a few changes and enhancements to CRTP in ECRTP. First the new packet formats for Compressed UDP and Full Header. A new checksum, HDRCHKSUM, is included in ECRTP to replace a zero UDP checksum. Functionality to be able to repeat changes in several subsequent packets is also added. The changes and associated issues are discussed in the sections below.

3.1.1 Packets Formats

The packet format Full Header had some changes. The first two length-fields in the IP/UDP/RTP header and possible encapsulating headers are changed in the ECRTP compressor. The fields are used to send information about the flow to the decompressor. In CRTP, the compressor sends a context identifier, a sequence number and a generation number. In ECRTP, the same fields are sent but the formats of the fields are changed. The changes are described in section 1.2.8 and 1.2.10. The C bit is included in the length fields indicating the new header checksum, replacing the missing UDP checksum over the compressed link. A check for a zero UDP checksum when parsing through the whole header must be done.

The Compressed UDP packet format is changed. Delta values can be in both IP and RTP header. The format does not only compress encapsulating headers and IP/UDP, but can compress the RTP header too. The Compressed UDP packet format is extended to be able to send both delta values and absolute values for the DELTA fields.

3.1.2 Determine Packet Type

In Both CRTP and ECRTP the compressor context for the incoming packet identified by the IP source and destination addresses, UDP source and destination port and RTP SSRC field. Other fields like; IPv4 Type of Service, IPv4 TTL, IPv6 Hop Limit and IPv6 Flow Label can also be included and used for identification. Parsing through all encapsulating headers is necessary to identify the innermost IP header and to verify that all encapsulating headers are of the same type and order as in the context. If no

matching context is found a new context must be created. Contexts are recycled by least-recently-used principle when all contexts are used.

When the context is identified, the compressor should pick the most advantageous packet type to be sent. There are two independent tasks that determine the packet type to be sent. Comparing the packet with the context and checking if a repetition sequence of is in progress. Comparing the packet against the context is done the same way as in CRTP. Fields that have changed are identified and second order changes for delta fields are checked.

Further, ECRTTP must send updates to the decompressor context at least N+1 times. Updates are only sent in Full Header and Compressed UDP packets.

Consequently, if the comparison against the context allows sending a Compressed RTP, the repetition sequence check can override the decision and a Full Header or Compressed UDP should be sent. The other way around is possible as well. If the compressor is in a Compressed UDP repetition send sequence, then the context comparison check can force the compressor to send a Full Header. This way of choosing packet type is the most aggressive compression for maximizing the compression rate that is allowed. The compressor can of course always choose a larger packet type if it find that more favourable.

Three different types of packet type selector are implemented in the compressor:

NORMAL	Always tries to select the smallest packet type.
NODELTA	Never choose Compressed RTP. Always encode IPv4 Identification, RTP Timestamp, RTP Sequence Number as absolute value and not as delta (Only for IPv4).
NODELTA IPv4 ID	Never choose Compressed RTP. Always encode IPv4 Identification as absolute value and not as delta (Only for IPv4).

3.1.3 N Repetition in the Decompressor

The decompressor context needs to be extended as well. To be able to derive N, from the initial N+1 Full Header sequence, the decompressor needs a counter. It also needs a counter when sending CONTEXT STATE to reestablish the contexts.

3.1.4 Checksum and TWICE

The implementation of the new header checksum has no serious issues associated to it. However, there is an issue that is not obvious at first inspection. There can be an authentication header between the IP header and the UDP. Caution should be exercised when implementing the checksum functionality. The pseudo header is the same for an IP/AH/UDP/RTP as for IP/UDP/RTP for checksum calculations. The protocol or next header field in IPv4 or IPv6 will always be UDP for checksum purposes, even if there exists header in between.

RFC3545 gives room for TWICE even if more than N packets are lost. In this implementation the maximum link sequence number gap can be specified at compile time. If the compressor has a large computational load it should be configured to a low value. The success rate decreases rapidly for every additional try to use TWICE.

For IPv4 however when using TWICE to recover for more than N lost packet should not allowed. The IPv4 Identification is not protected by the UDP checksum and an erroneous packet can be created if TWICE for more than N is allowed. The decompressor in this implementation does not TWICE for IPv4 if more than N packet is lost and IPv4 Identification is not sent as an absolute value.

3.1.5 Decompressor Feedback

To avoid flooding the return channel between the decompressor and the compressor, some restrictions must be imposed on the decompressor. Otherwise, the N+1 context state packet sent for each reconstruction failure will flood the reverse channel at times of heavy packet loss for a decompressor with many simultaneous connections. The restriction should be per decompressor and not per context. The decompressor must have a timeout and send another N+1 context state packet if the decompressor still is out of sync. This implementation does not implement these features. The decompressor sends N+1 feedback per failed reconstruction instead. It gives a very good performance when running only a few flows but does not scale well for a decompressor with many active simultaneous contexts. Normally, the N+1 header requests are sent a few seconds apart, several flows can be sent in the same packet. The implemented solution however, will immediately send N+1 header requests for each failed incoming packet. This makes it vulnerable for denial of service attacks and at times of bad link conditions. Every incoming malicious packet or packets belonging to a damaged context, causes N+1 packet to be sent to the compressor.

3.1.6 Classifying RTP and Negative Cache

It is very difficult to classify IP/UDP/RTP packets, since the UDP header does not include a next protocol field. Heuristics have to be used to determine if a packet is an RTP packet. Since the packets associated to a context using the RTP SSRC, single source field, there can be problems when trying to compress IP/UDP packet. What is expected to be the SSRC field is actually a part of the payload. The SSRC field will change more frequently than for a real IP/UDP/RTP packet allocating several contexts. This can lead to a context thrashing when allocating a new context for almost every arriving packet. The fast allocating of context can force real RTP out of the context space. The RTP flow has to restart the compression more regularly and compression efficiency will be reduced. It is even more important for ECRTP than CRTP, since the initial context synchronization in ECRTP uses more redundancy.

To solve this problem, flows with changes in the constant RTP fields, is placed in a negative cache. These flows are not allowed to change context until they behave like a normal flow. They will be compressed with Compressed UDP traditional or Full Header when in the negative cache. When the flow is behaving like an RTP flow, it is removed from the negative cache and can be compressed as an RTP flow again. The implementation used in this master's thesis deploys only a simple implementation of the negative cache. When a flow has been put in the negative cache it stays there until the context is recycled.

3.1.7 Checksums

The ECRTP algorithm uses a checksum to validate packets and to detect packet loss or reordering and to recover from those problems (i.e. TWICE algorithm).

Since the checksum is a summation of some of the fields in the headers it can be reused when using TWICE. This can reduce the computational load in the decompressor. However, if the HDRCHKSUM is used for an IPv4/UDP/RTP flow, the checksum calculation can be reused even between packets.

Example: HDRCHKSUM reused between packets. Checksum are one-complement to the real checksum. The actual checksum is within parenthesis.

```
Context :
HDR checksum           = 0x3430 (0xcbcf)
Payload Type           = 0x00
Delta ipv4 Identification = 0x1
Delta RTP Timestamp    = 0x160
Delta RTP Sequence Number = 0x1
```

A Compressed RTP packet, with proceeding link sequence number, arrives:

```
Compressed header packet:
HDR checksum           = 0x3593 (0xca6c)
Payload Type           = 0x01
```

Add Delta values and new Payload Type to old checksum.

```
(one-complement to new checksum) =
  (old checksum) + (Delta IPv4 Identification) +
  (Delta RTP Sequence Number) + (Delta RTP Timestamp) +
  (new Payload Type)-(old Payload Type))
```

```
(one-complement to new checksum) =
  0x3430 + 0x1 + 0x1 + 0x160 + (0x01-0x00) = 0x3593
```

The new calculated checksum is one-complement to 0x3593 which is equal to the incoming checksum of the packet. This feature is not implemented in the prototype ECRTP implementation used in the simulations.

Chapter 4

Simulations

4.1 Simulations

In this section the simulation setup will be described. Simulation scenarios and metrics will be described and explained. Implementation details and configurations of importance will be discussed.

4.1.1 Effnet HC-Sim

Effnet HC-Sim is used to test Effnet's header compression products under varying link conditions. Effnet HC-Sim includes functionality to manipulate the packets before and after the compression. It provides means to lose packets before and after the compression. Functionality was added to be able to reorder packets between the compressor and the decompressor.

The following properties of the link are manipulated during the simulations:

- Link delay
- Link packet reordering frequency
- Link packet loss frequency

Compression Algorithms

To evaluate ECRTP and put the result in context, it is compared to other compression algorithms that are currently in use. The choice of compression algorithms to compare ECRTP with, fell on CRTP and ROHC. ECRTP is an extension to CRTP and a comparison will answer if extensions make ECRTP live up to its prerequisites stated in the RFC3545. ROHC was chosen because the RFC3545 mentions ROHC as an additional candidate besides CRTP, to be extended to handle packet reordering, packet loss and large delays. ROHC uses LSB encoding which is an interesting compression technique for packet reordering on the channel between compressor and decompressor. Another reason is that Effnet AB has complete implementations of both CRTP and ROHC.

Header compression algorithms used in simulations are:

- ECRTP
- CRTP

- ROHC

Some of the schemes can be configured to alter their behaviour. ECRTP can be configured in several ways. The repetition value (N) will be set to a value between zero and nine. As mentioned in section 3.1.2 the ECRTP compressor has been implemented in three different modes: Normal mode, NO DELTA and NO DELTA IPv4 ID. Normal mode will use TWICE if sequence number gap is smaller than N+1 for IPv4. NO DELTA send only absolute values and will not use TWICE and NO DELTA IPv4 ID will try to TWICE RTP Sequence Number and RTP Timestamp three times before failing.

CRTTP will be configured to use TWICE only once and invalidate the context when failing to reconstruct a packet in the decompressor.

ROHC have three modes: Unidirectional mode, Bidirectional Optimistic mode and Bidirectional Reliable mode. The modes will be denoted as U-mode, O-mode and R-mode. O-mode and U-mode have a configurable repetition value (L). It is similar to ECRTP N value, and specifies how many times the context establishing packets should be sent. U-mode uses only positive acknowledgment in the simulations. The CID space will be between 0 and 15, encoded as four bits for all ROHC modes.

4.1.2 Simulation Setup

Header compression is designed to be deployed point-to-point, over one link or over a tunnel. Different link layers or tunnels are not simulated. Instead, the network between the compressor and the decompressor are represented by three attributes: packet loss probability, reorder probability and delay. These properties can be set in both directions. Consequently, the delay can be used to simulate an asymmetric link or an asymmetric tunnel through a network.

Figure 4.1 is a schematic view of the simulation setup for Effnet HC-Sim. RTP

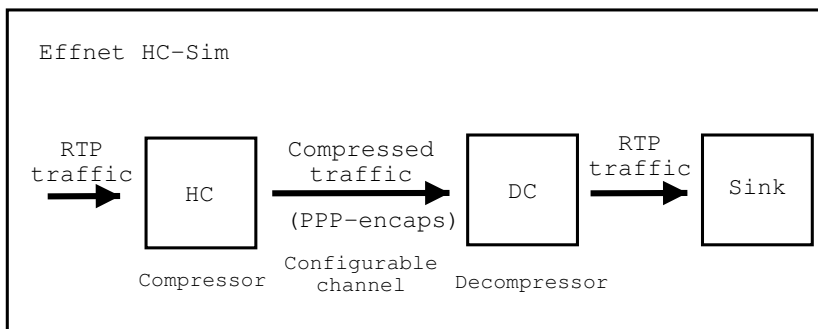


Figure 4.1: Effnet HC-Sim Simulation Setup

traffic is fed into the Effnet HC-Sim, either from file or from Effnet HC-Sim packet generator. The Compressor compresses and encapsulates the packet into PPP and sends it through the configurable channel or network to the decompressor. The decompressor reconstructs the packet and passes it on to a sink. The sink gathers statistics and performs error checking on the reconstructed packets.

4.1.3 Sample Traffic

Realistic RTP traffic was gathered by initiating an RTP conference using RAT[13], robust audio tool, and passing the packets through the Effnet HC-Sim tool. The IP/UDP/RTP traffic was recorded in Effnet HC-Sim. The recorded traffic can then be fed into the Effnet HC-Sim tool and test if the different header compression schemes can be performed under varying link and network conditions.

A schematic view of RTP capture procedure is shown in figure 4.2. The sending hosts routing table was changed to the host with Effnet HC-Sim. Effnet HC-Sim was configured to pass on the traffic after running it through the simulated link, by changing the link layer address to the receiving host. HC-Sim normal logging features are used writing the packets incoming to file.

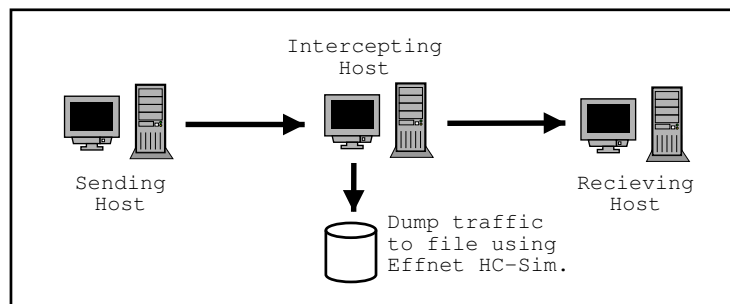


Figure 4.2: Capturing RTP traffic by intercepting Ethernet traffic

To avoid dependency to the payload encoding in the simulation several codec were used. The codec used were GSM, G7.11 A-law, G7.11 my-law and linear 16-bit audio codec.

4.1.4 Metrics

The choice of the scenarios is constructed to point out differences between the header compression schemes. The metrics for all the scenarios is compression efficiency, decompression errors and decompression failures.

The compression efficiency is the number of bytes removed from the header compared to the uncompressed header, no payload included. Decompression error is number of packets that are reconstructed incorrectly. The packets are checked and compared by the simulator before the compression and after the decompression or reconstruction. Decompression failure is the number of packets that the decompressor was unable to reconstruct, due to unsynchronized contexts or a delayed packet belonging to an earlier context. The decompressor discards all failed packets.

Comparing the compression efficiency makes sense since the goal with header compression is to avoid sending redundant data. All header compression schemes have preconceived ideas about the traffic and sometimes packets are lost or incorrectly reconstructed by the compression scheme. It is important to remember when dealing with header compression that you are a part of the network. The network should preferably not make those decisions. It is up to the application to make the decision of which packet that can be discarded or not. That is why the UDP checksum is sent “as-is” in all the compression algorithms. This is also the reason why minimizing the compres-

sion related packet error and packet loss is important and mandates the measurement of reconstruction errors and reconstruction failures.

Simulation Scenarios

The simulation scenarios are chosen to see if ECRTP lives up to the properties specified in RFC3545 and to discover and show possible weakness and properties of ECRTP. Another goal is to find scenarios where the algorithms differ from each other.

Simulation scenarios:

- IPv4/UDP/RTP audio playback stream.
- Generated IPv4/UDP/RTP with prelink reordering.
- Generated with changes in checksum-protected fields.

All simulation scenarios of RTP traffic types are tested under different link conditions. The algorithms are simulated under different configurations over links with different packet reordering frequencies and packet loss frequencies and delay. Packets that are reordered are delayed in time using a random number generator. They will be reordered between 0-15 packets, depending on the packet send rate of the codec. Each of the test cases was feed with 20000-25000 packets approximately half an hour of traffic.

4.2 Simulation Results

This section present the simulation results. Many test cases were run but only a few test cases for each simulation are presented and analyzed here. For algorithms with a repetition value, only simulation cases with one value are accounted for. The repetition value of three has been shown to perform well for both ROHC and ECRTP. N equals two for ECRTP and L equals three corresponds to a ROHC repetition value of three. Two delays were chosen, 15 ms and 105 ms, to represent a long and a short round-trip-time.

4.2.1 Audio Stream Simulation

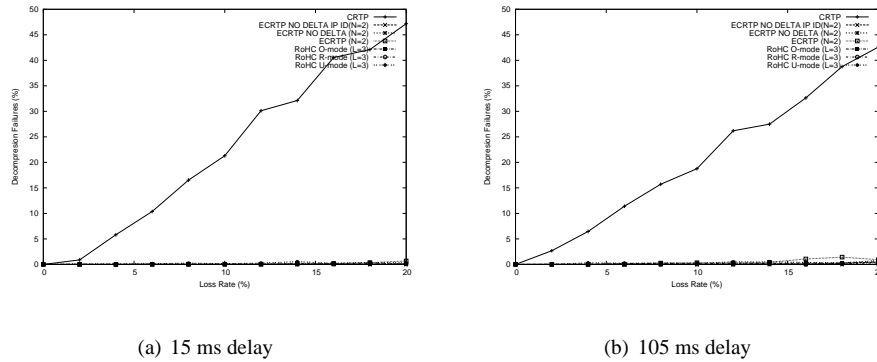
This section presents and analyzes the audio stream simulation scenario. The IPv4 audio streams are characterized by monotonically increasing RTP timestamps and RTP Sequence Numbers between subsequent packets. The IPv4 Identification is increased by one on a “per stack” basis and will often be increased by two or three because of other IP traffic on the sending host. The sending application is using silence suppression that will cause unpredictable changes in both IPv4 Identification field and RTP Timestamp field.

Audio Streams and Packet Loss

Most of the header compression schemes in the simulations are constructed to be able to handle at least some packet loss. Figure 4.3(a) and 4.3(b) shows the decompression failure rate for the different compression schemes.

At 15 ms delay (figure 4.3(a)), most of the compression schemes handle packet loss rather well, except CRTP. The decompression failure rate increases linearly with

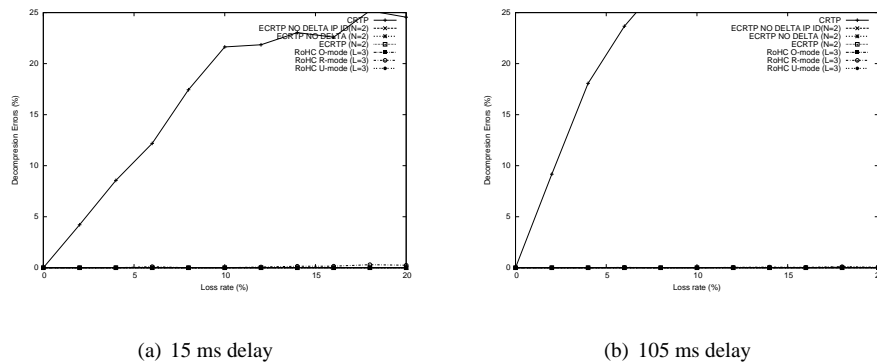
Figure 4.3: Failure Rate during Packet loss



the packet loss rate for CRTP. ECRTTP in normal mode starts losing a few percent of packets at very high packet loss frequency. ECRTTP and ROHC, independent of mode, seems to handle packet loss over links with small round trip time very well. CRTP does not handle packet loss very well. The use of TWICE is not enough to handle packet loss.

Figure 4.3(b) shows the failure rate for packet loss over links with 105 ms delay. The graph is almost identical to the figure 4.3(a). CRTP distinguish itself from the rest by losing packet in a linear manner with the packet loss rate. Surprisingly, the delay does not seem to have any impact on the reconstruction failures for any of the compression algorithm in this simulation scenario.

Figure 4.4: Error Rate during Packet Loss



The reconstruction error rate at 15 ms delay is shown in figure 4.4(a) and at 105 ms delay in figure 4.4(b). Again, CRTP is distinguished from the rest by creating more erroneous packets. It does not seem to be able to handle the changes in the IPv4 Identification fields. The repetition of updates, used by ECRTTP and ROHC seem to achieve a more robust operation during periods of packet loss, even for packets that are not protected by the UDP checksum. ROHC R-mode seems to reconstruct a few percentages the packet incorrectly at very high rates of packet loss. This is probably a result of

R-modes sparse use of protecting CRC, compared to O-mode and U-mode.

In figure 4.4(b) at delay 105 ms, CRTP reconstructs even more broken packet for the higher delay over links with packet loss. R-mode has fewer errors when the round trip time increases, than the other ROHC modes. This is probably because R-mode waits for acknowledgement on context updates before it uses the new context to compress. Longer round trip time allows fewer context refreshes, due to rate limits on header requests, and increases the number of possible errors.

Figure 4.5: Compression Ratio during Packet Loss

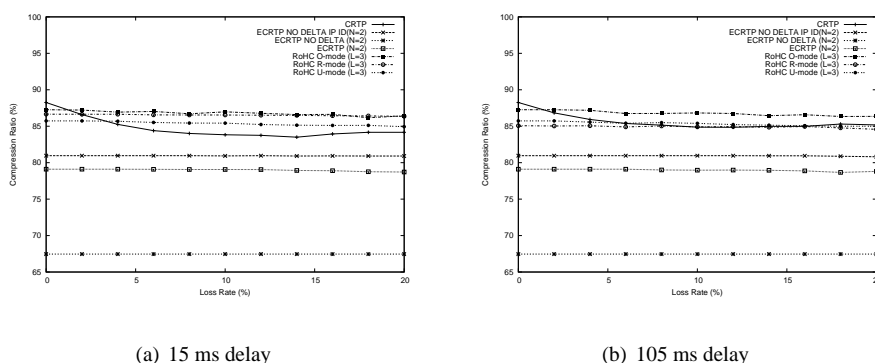


Figure 4.5(a) and 4.5(b) shows the compression ratio over links with different packet loss frequencies. The compression ratio does not decrease when packet loss increases for any of the compression algorithms except CRTP. CRTP decreases a few percent when packet loss increases. It decreases less if the round trip time is lower. This is reaction to smaller failure rates for higher delays seen in figure 4.5(a) and figure 4.5(b). More decompression failures will trigger a new context synchronization decreasing the efficiency of the compression.

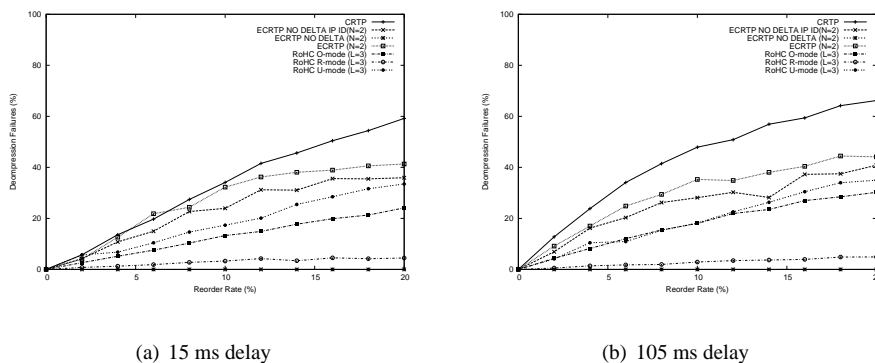
Audio Streams and Packet Reordering

This section will analyse and discuss the result from the simulation of audio streams over links with packet reordering. ECRTTP is the only compression scheme that discusses the subject of reordering.

Figure 4.6(a) and figure 4.6(b) shows the decompression failure rate for during reordering. CRTP loses many more packets than the number that is reordered. ECRTTP and ECRTTP NODELTA IP ID and ROHC U-mode and O-mode fail to decompress a significant fraction of packets. For every reordered packet CRTP fails to decompress three packets. ECRTTP and ECRTTP NODELTA IP ID is not dependent of the delay, while CRTP and ROHC O-mode are. ECRTTP NODELTA is handling reordering of the audio perfect. It does not lose any packets at all. ROHC R-mode performs well under these conditions. R-mode fails to decompress about every fourth of the reordered packets. ECRTTP NODELTA has no decompression failures for the audio streams at all.

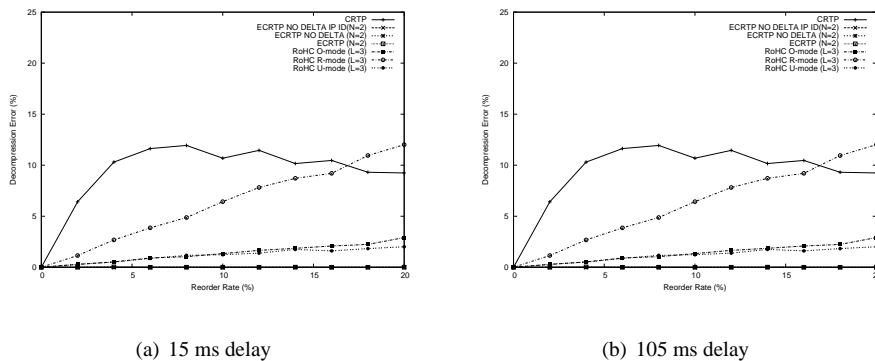
In figure 4.7(a) and figure 4.7(b) the error rate during reordering can be found. CRTP produces a lot of incorrect packets. The CRTP curve is rising steeply at low reorder rates and flattens out and even decreases for higher rates. This is probably because CRTP reaches a point where all possibilities to get reconstruction errors are ex-

Figure 4.6: Failure Rate during Packet Reordering



plotted. The effect of the many decompression failures can be observed in figure 4.6(a) and (b). The decompressor context becomes invalid and packets are automatically dropped without trying to reconstruct the packet. Fewer reconstructions will lead to fewer errors. A larger delay adds to the number of errors since the time to reestablish is longer after the next decompressor failure.

Figure 4.7: Error Rate during Packet Reordering

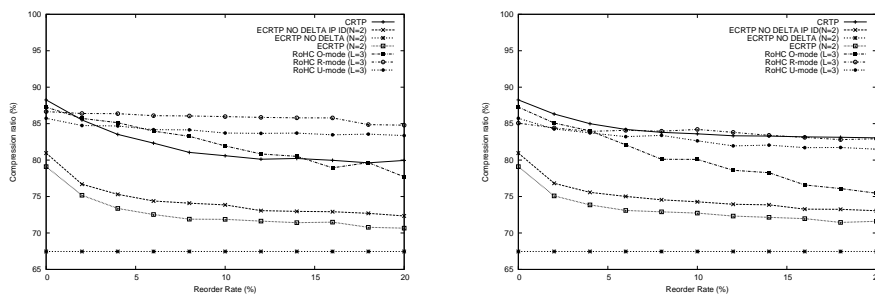


ROHC R-mode has a large error rate for reordering compared to the other ROHC modes. In R-mode, the usage of CRC is less common than in the other modes, and therefore error checking is not as good as for the other ROHC modes. R-mode reconstructs fewer incorrect packets at longer delays. The delay adds to the time before the compressor receives feedback, and the interval between updates to the context becomes longer. With fewer updates to the context, the chance of reordering a context update decreases, and the error rate decreases.

A very small fraction of the packets are incorrectly reconstructed by ECRTMP, while ECRTMP NODELTA and ECRTMP NO DELTA IP ID does not create any erroneous packets at all.

Figure 4.8(a) shows how CRTP, ROHC U-mode, ECRTMP and ECRTMP NODELTA are affected by the change of reorder frequency. All schemes decrease the compression

Figure 4.8: Compression Ratio during Packet Reordering



(a) 15 ms delay

(b) 105 ms delay

ratio when the reorder frequency increases. This is actually the result on the increase of decompression failures rather the reorder rate. The compression algorithms that have few decompression failures have an almost constant compression ratio independent of the reorder rate on the link.

When the delay increases (figure 4.8(b)) the compression rate seem to become less dependent of the reorder frequency. ROHC O-mode is an exception, were dependency increases with the delay. The compression ratio is coupled to failure rate. This phenomenon can be studied in figure 4.7 and figure 4.6.

4.2.2 Audio Stream Simulation Summary

Summary of the different compression schemes for audio stream with links with high packet loss frequencies or high reorder frequencies.

Algorithm	delay 15 ms (%)			delay 105 ms (%)		
	compression	failures	errors	compression	failures	errors
CRTP	83.78	19.20	21.63	85.00	17.85	31.53
ECRTP	79.06	0.04	0.00	78.97	0.30	0.00
ECRTP NO DELTA IPv4	80.93	0.02	0.00	80.96	0.00	0.00
ECRTP NO DELTA	67.46	0.00	0.00	67.46	0.00	0.00
ROHC O-mode	86.98	0.03	0.00	86.83	0.22	0.00
ROHC U-mode	85.43	0.13	0.00	85.39	0.10	0.00
ROHC R-mode	86.54	0.01	0.00	84.91	0.07	0.00

Table 4.1: Packet loss simulation result for CRTP, ECRTP and ROHC at average loss of 10 %

All compression schemes except CRTP seem to handle packets loss in the audio streams quite well. ROHC have the most effective compression of the algorithms.

Studying the table 4.1 gives some information about the audio streams. IPv4 Identification is changes often enough to make absolute encoding more efficient then delta coding in ECRTP. ECRTP is less efficient than ECRTP NO DELTA IP ID.

CRTP handling of packet reordering is not good as can be seen in table 4.2. ECRTP and ECRTP NO DELTA IP ID is better but they still fail to decompress a lot of packets.

Algorithm	delay 15 ms (%)			delay 105 ms (%)		
	compression	failures	errors	compression	failures	errors
CRTP	80.61	34.22	7.38	83.52	46.66	12.41
ECRTP	71.87	32.15	0.00	72.73	35.15	0.00
ECRTP NO DELTA IPv4	73.85	23.87	0.00	74.28	28.08	0.00
ECRTP NO DELTA	67.46	0.00	0.00	67.46	0.00	0.00
ROHC O-mode	83.71	17.34	1.13	80.09	18.01	1.34
ROHC U-mode	81.94	13.27	1.56	82.63	18.19	1.23
ROHC R-mode	85.97	3.27	6.47	84.20	2.90	6.43

Table 4.2: Reorder simulation result for CRTP, ECRTP and ROHC at average reordering of 10 %

ROHC O-mode and U-mode handles reordering little bit better. ROHC R-mode handles reordering fairly well, but the decompressor redistributes many erroneous packets. However, ECRTP NO DELTA handles reordering very well, but at the cost of compression efficiency. ECRTP NO DELTA has a significantly lower compression ratio than the other schemes. ROHC and CRTP have the highest compression ratio.

One conclusion that can be drawn from this simulation scenario is that the delta encoding used in ECRTP should not be used on links with reordering. ROHC and ECRTP NODELTA perform better than the other schemes and they both use some kind of absolute encoding.

4.2.3 Prelink Reordering

In section 2.1.8 in page 19, reordering before the compressor is discussed. ECRTP is very sensitive to unexpected changes in the flow since it is trying to find a relation between the link sequence number and the DELTA fields. ROHC is using another strategy. ROHC tries to find a relation between the RTP Sequence Number and the other DELTA fields. If the packets are reordered before the decompressor the possible connection between the link sequence number is broken, while the connection to the RTP Sequence Number still could be present.

To show the weakness of the strategy used by ECRTP a test case is added. First, a test case is run with very predictable RTP flow, with monotonically increasing DELTA fields and some silent periods. After that the same stream is simulated again, but this time the flow is reordered before the compressor. Every twentieth packet is moved sent after the nineteenth packet.

Algorithm	Predictable stream (%)	Reordered stream (%)
CRTP	90.00	87.50
ECRTP	90.00	81.00
ECRTP NO DELTA IPv4	82.50	76.62
ECRTP NO DELTA	67.50	67.49
ROHC O-mode	90.00	90.00
ROHC R-mode	90.00	90.00
ROHC U-mode	87.80	87.80

Table 4.3: compression ratio for CRTP, ECRTP and ROHC with and without prelink reordering

(%) As can be seen in table 4.3, ECRTP is compressing 9 percent less efficient and CRTP about 10 percent for the reordered stream, while ROHC and ECRTP NO DELTA have the same compression ratio for both streams.

This is maybe not a realistic RTP flow, but the test case clearly shows that ROHC is more efficient under the presence of prelink reordering than ECRTP and CRTP. The compression of ECRTP NODELTA is sensitive to prelink reordering, but it has a significantly lower compression rate.

Algorithm	Predictable stream (%)			Reordered stream (%)		
	normal	loss	reorder	normal	loss	reorder
CRTP	0.00	43.48	66.98	0.00	59.03	67.58
ECRTP	0.00	40.63	64.28	0.00	2.23	25.19
ECRTP NO DELTA IPv4 ID	0.00	0.00	8.21	0.00	0.01	11.40
ECRTP NO DELTA	0.00	0.00	0.00	0.00	0.00	0.00
ROHC O-mode	0.00	0.00	9.15	0.00	0.00	9.08
ROHC R-mode	0.00	0.00	0.10	0.00	0.00	0.15
ROHC U-mode	0.00	0.00	49.52	0.00	0.00	49.46

Table 4.4: Failure rate for CRTP, ECRTP(N=02) and ROHC(L=3) with and without prelink reordering for links with 10% loss, link with 10% reorder and link without reordering and loss.

Generally, the more context updates sent from the compressor to the decompressor the more vulnerable it becomes to reordering and loss on the link between them. In this case, ECRTP becomes more stable against reordering and loss of the link. It starts sending the absolute values more often and becomes more robust. The repetition of Full Header and Compressed UDP packets has a large impact on the result on both the robustness and the compression ratio of ECRTP. The result of the strategy can be seen in table 4.4 and table 4.3. It becomes more robust, but the compression ratio increases.

Algorithm	Predictable stream (%)			Reordered stream (%)		
	normal	loss	reorder	normal	loss	reorder
CRTP	0.00	0.00	0.00	0.00	0.00	0.00
ECRTP (N=2)	0.00	0.00	0.00	0.00	0.00	0.00
ECRTP NO DELTA IPv4 ID	0.00	0.00	0.00	0.00	0.00	0.00
ECRTP NO DELTA	0.00	0.00	0.00	0.00	0.00	0.00
ROHC O-mode	0.00	0.00	1.41	0.00	0.00	1.34
ROHC R-mode	0.00	0.00	0.95	0.00	0.00	0.96
ROHC U-mode	0.00	0.00	0.67	0.00	0.00	0.66

Table 4.5: Error rate for CRTP, ECRTP and ROHC with and without prelink reordering for links with 10% loss, link with 10% reorder and link without reordering and loss.

Table 4.5 shows the number erroneous headers the decompressor reconstructs. ROHC creates more erroneous header after decompression compared to ECRTP. There is difference in table 4.5 compared to the result in audio streams in section 4.2.2. ROHC O-mode creates more erroneous packets than R-mode.

4.2.4 Changes in Checksum-Protected Fields

Simulations with changes in UDP checksum-protected fields. ECRTP and CRTP use the checksum to validate reconstruction of packets in the decompressor. ROHC uses a cyclic redundancy check (CRC) to validate the construction of changed fields. The field changed in simulation is CRSC list and Payload Type. Other possible changes could be RTP Padding and RTP Extension changes, but these fields are not changed in this simulation run.

There are a few realistic cases where these fields could change. Both RTP fields are normally constant throughout an RTP session, but there are codecs that adapt to network environment where the Payload Type could change within the same RTP stream. It has been mentioned earlier that several RTP streams can be multiplexed into one stream. Multiplexing can be used with success when tunnelling multicast traffic through a firewall. The CSRC list is used to keep track of the contributing sources of the multiplexed packet. The CSRC list may vary as the different multicast participants send traffic to the multicast group.

Figure 4.9: Changes in Checksum-Protected Fields over Links with Packet Loss

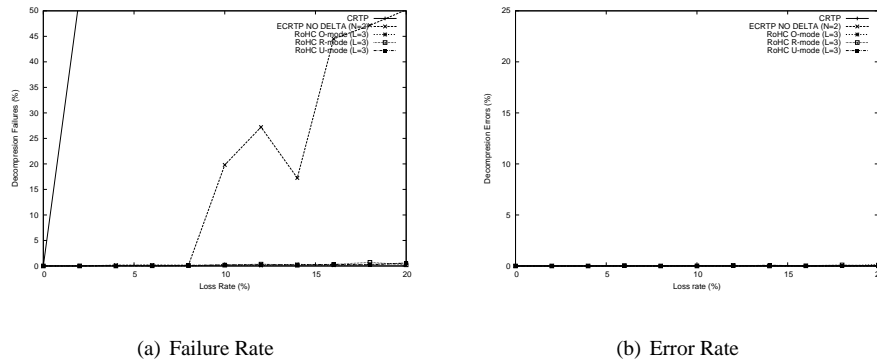


Figure 4.10(a) shows the decompression failure rate and figure 4.10(b) shows the decompression error rate, both during packet loss. CRTP have no functionality to increase robustness for packet loss for changes in these RTP fields. Figure 4.10(a) shows how the decompression failure rate increases rapidly when packet loss increases. ECRTP seem to handle the packet loss to some extent. ECRTP handles all losses of less than three packets in a row. In this simulation, loss bursts of three or more is not present for loss frequencies less than eight percent. The N value has a large impact for how ECRTP handles losses when changes in checksum-protected fields occur. ROHC seem to handle packet loss for these fields very well.

In Figure 4.11(a) and figure 4.11(b) the failure rate and error rate during reordering is showed. In figure 4.11(a), the repetition of context updates seems to have a limited effect on robustness against reordering. ECRTP repeats context updates and has only a few percentage better failure rate than CRTP that does not repeat context updates. ROHC handles reordering better and fails the reconstruction of one out of two packet. In figure 4.11(b) it can be seen that all the compression algorithms reconstructs a small fraction of incorrect packets during reordering.

Figure 4.12(a) and figure 4.12(b) shows the compression ratio during packet loss and reordering. Again we can see the effect of the many decompression failure for

Figure 4.10: Changes in Checksum-Protected Fields over Links with Packet Reordering

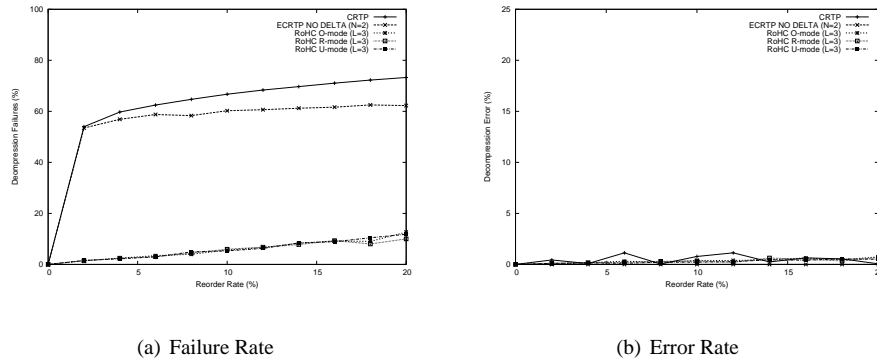
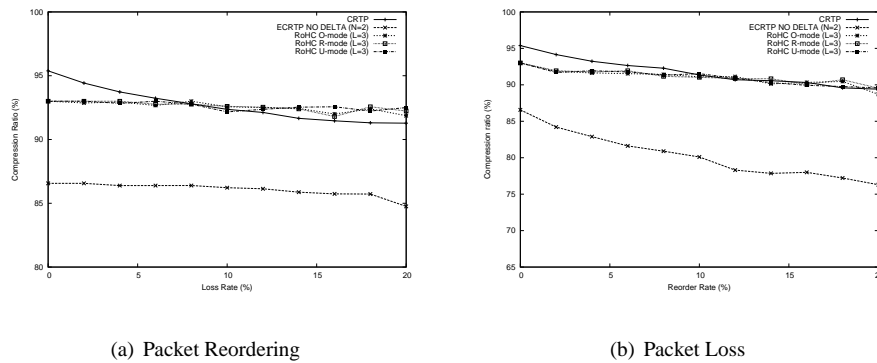


Figure 4.11: Compression Ratio for Changes in Checksum-Protected Fields



CRTP and ECRTP during reordering. The compression ratio drops steeper for ECRTP and CRTP for reordering than for ROHC.

4.3 Simulation Summary

All the simulation scenarios show that ECRTP work well over links with packet loss. The round-trip-time seems to have very little influence on the result of the simulations for packet loss. The repetition value and the absolute establishment seem to improve ECRTP's ability to handle packet loss compared to CRTP.

The prelink reorder simulation scenario show that absolute encoding can be more efficient than delta encoding, for frequently changes field with monotonically predictable changes, during prelink loss or reorder.

Reordering can be handled by sending only Compressed UDP packets and absolute encode the DELTA fields. ROHC shows good result for reordering show that LSB encoding could be a more efficient solution to compress the DELTA fields rather than the more redundant absolute encoding.

The simulations show that repeating the context updates is enough to handle link

reordering. Most updates are established correctly in the decompressor context. However, the decompressor still fails to reconstruct these fields for delayed packets. Previous updates can arrive after later updates, and the contexts will be put out of synch. The decompressor need to be able to protect itself from late arriving packet, derived from a surpassed context. The simulation scenario with changes in checksum-protected fields show these fields should be encoded using some kind of absolute encoding or at least be sent as is in the compressed packet. The list compression type 0 of CSRC list in ROHC is more successful than handling of CSRC list in EC RTP.

Chapter 5

Conclusion

5.1 Conclusion

This section concludes the result of the evaluation. If ECRTP lives up to the goals stated in the RFC3545. How ECRTP be configured to handle packet loss and packet reordering. Suggested improvements and other interesting result are summarized.

5.1.1 The Prerequisites of ECRTP

The assumptions of the ECRTP, described in RFC3545, are that CRTP does not handle reordering and packet loss well. That long round-trip-times makes the compression schemes perform worse over links with packet reordering and high packet loss frequencies.

Simulations and the theoretical investigation in this thesis show and explain that the above assumptions are correct. In the simulation, CRTP does handle neither packet loss nor packet reordering well. There were significantly more decompression failures and decompression errors when the round-trip-time increased, over links with packet loss and over links with packets reordering.

One conclusion can be drawn from this. Extensions to CRTP are needed to handle such harsh link condition.

5.1.2 ECRTP Solutions

The idea behind the solution in ECRTP is to try to avoid losing the synchronization between the compressor and decompressor context. This is done by using absolute encoding instead of delta encoding and by repeating all context updates several times. Compressor feedback is only used to recover from serious failures, and not primarily used to keep the contexts synchronized. The authors of the ECRTP specification wanted to make as few changes as possible from CRTP and to keep it as simple as possible.

Avoiding loss of synchronization is an attractive strategy for several reasons. The scheme can easily be adapted to work well on unidirectional links and for multicast and broadcast traffic. The scheme also becomes less dependent on the round-trip-time. The drawback of this strategi is decreased compression ratio as a result from sending more redundant packets.

The delta encoding used by ECRTP does not perform well over links that reorder packets. For header fields that are changing frequently absolute encoding should be

used instead of delta encoding. The ECRTP compressor should use some kind of absolute encoding to avoid incorrect reconstruction of monotonic delta values, such as IPv4 Identification, RTP Timestamp and RTP Sequence Number. LSB encoding can be used instead of the absolute encoding described RFC3545.

Context update repetition is used to avoid loss of context synchronization. Context updates sent from compressor to decompressor are always vulnerable on link that reorder packets. It can be seen in the simulation of ECRTP that repetition of context updates decreases the number of context failures over links with packet loss and reordering.

Other fields that changes less frequently in the RTP header should be sent “as is” in the compressed packet. For applications that tries to adapt to the network bandwidth by changes the codec, the Payload Type should be send be sent in all compressed packet during reordering. When RTP traffic is multiplexed, the CSRC list should always be sent in every packet or with a high repetition value. An improvement to ECRTP, would be to send only the index of contributing sources of the list.

IETF RFC3545 has a logical error in the specification. It defines a value link quality value N . It should be set so that it is a very small probability that N consecutive packet is lost over the link. RFC3545 stresses that if every context update sent from compressor to the decompressor are repeated $N+1$ times, the context will be in sync as long as fewer than N packets are lost. N is also used in the decompressor to send $N+1$ header requests in order to guarantee delivery. $N+1$ is distributed from compressor to decompressor by sending $N+1$ Full Header at the beginning of the session. If packets are lost the decompressor can get a smaller value and a sequence of header requests can be lost even if less than N packets are lost.

The specification only implies how reordering should be solved. A more detailed implementation guide could complement the specification

In this thesis two other solutions to extend ECRTP is presented. These will both improve the compression rate compared to ECRTP NODELTA but will not handle reordering and loss as well as ECRTP NODELTA. The negative TWICE strategy is the most efficient but will probably lose more packets during reordering. It will also perform less efficient if reordering occurs before the compressor. The relative encoding is less vulnerable because it sends less context updates but will compress less efficient than negative TWICE. However the simulations of checksum-protected fields shows that establishing new values of fields that changes frequently are vulnerable to reordering.

ECRTP implemented in the straightforward way from the specification performs well over link with packet loss, but it does not improve CRTP over links with reorder significantly. However, ECRTP can within the specification be made to work well over tunnels and links of with reordering and high packets loss frequencies. With a not too small N and the NODELTA configuration, ECRTP performs well in such environments. However if Payload Type or CSRC-list changes frequently the repetition value should be larger or the fields should be sent as is.

The ECRTP header compression scheme handles packet loss well and can be configured to handle reordering well. All header compression schemes discussed in this master’s thesis can be destructive. Introducing header compression in a network can increase the number of erroneous packets passed on. However, the bandwidth usage can significantly decrease for some types of traffic where header compression is used.

5.2 Future Work

The ability to handle packet loss is solved by both ROHC and ECRTP. Absolute or LSB encoding should be used for header fields that are monotonically changing over links that reorder, but there are other fields that lack a good strategy for efficient compression during reordering. How to compress the other RTP fields, such as Payload Type and the CSRC-list needs to be further investigated. Perhaps ROHC is a better candidate for extension to handle link reordering than CRTP.

Bibliography

- [1] Koren T, Casner S, Geevarghese J, Thomson B, Ruddy P: *Enhanced Compressed RTP for links with high Delay Packet Loss and Reordering*, RFC 3545, July 2003.
- [2] Mikael Degermark, Hans Hannu, Lars-Erik Jonsson, and Krister Svanbro: *Evaluation of CRTP Performance over Cellular Radio Links*, IEEE Personal Communications, August 2000.
- [3] Bruce Thompson, Tmima Koren and Dan Wing: *Tunneling Multiplexed Compressed RTP ("TCRTP")*, draft-ietf-avt-tcrtp-07.txt, November 2, 2002
- [4] Jerry Ash, Bur Goode, Jim Hand and Raymond Zhang: *Requirements for Header Compression over MPLS*, draft-ietf-avt-hc-mpls-reqs-00.txt, April 2004
- [5] Casner S, Jacobson V: *Compressing IP/UDP/RTP Headers for Low-Speed Serial Links*, RFC 2508, February 1999.
- [6] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC1889, January 1996
- [7] W. Simpson, Editor, *The Point-to-Point Protocol (PPP)*, RFC1661 , July 1994
- [8] C. Bormann, *Robust Header Compression (ROHC) over PPP*, RFC3241, April 2002
- [9] M. Engan, S. Casner, C. Bormann, T. Koren, *IP Header Compression over PPP*, RFC 3544, July 2003
- [10] Degermark M, Nordgren B and Pink S, *IP Header Compression*, RFC 2507 February 1999.
- [11] J. Postel, *User Datagram Protocol*, RFC 768, 28 August 1980
- [12] Bormann C, Burmeister C, Degermark M, Fukushima H, Hannu H, L-E. Jonsson, Hakenberg R, Koren T, Le K, Liu Z, Martensson A, Miyazaki A, Svanbro K, Wiebke T, Yoshimura T, Zheng H: *RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed*, RFC 3095, July 2001.
- [13] Orion Hodson, Colin Perkins: *Robust Audio Tool (RAT)*, Application found at <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/credits.html>, accessed 11 Jun, 2004.