Network Working Group
INTERNET-DRAFT
Expires: March 1, 2000

Lars-Erik Jonsson, Ericsson
Mikael Degermark, Lulea University
Hans Hannu, Ericsson
Krister Svanbro, Ericsson
Sweden
September 1, 1999

RObust Checksum-based header COmpression (ROCCO)
<draft-jonsson-robust-hc-01.ps>

Status of this memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or cite them other than as “work in progress”.

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/lid-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This document is an individual submission to the IETF. Comments should be directed to the authors.

Abstract

IP/UDP/RTP header compression [CRTP] can generate a large number of lost packets when used over links with significant error rates, especially when the round-trip time of the link is large.

This document describes a more robust header compression scheme. The scheme is adaptable to the characteristics of the link over which it is used and also to the properties of the packet streams it compresses. Robustness against link-loss is achieved without decreasing compression efficiency.

Table of contents

1. Introduction....4
2. Terminology....6
3. Existing header compression schemes....8
4. Desired improvements....9
5. Proposed solution....10
5.1. Base header format....11
5.2. Data structures....12
5.3. Header compression profiles....12
6. Classification of header fields....13
7. A header compression profile for IP-telephony packet streams.14
7.1. Usage scenarios, environment and requirements....14
7.2. Analysis of change patterns of header fields....15
7.2.1. IPv4 Identification....15
7.2.2. IP Traffic-Class / Type-Of-Service....16
7.2.3. IP Hop-Limit / Time-To-Live....16
7.2.4. UDP Checksum....17
7.2.5. RTP Marker....17
7.2.6. RTP Payload Type....17
7.2.7. RTP Sequence Number....17
7.2.8. RTP Timestamp....17
7.2.9. RTP Contributing Sources (CSRC)....18
7.2.10. RTP Timestamp delta....18
7.3. Packet types and code field usage....18
7.4. Robust encoding and decoding of delta information....19
7.4.1. Sequence number changes to handle....19
7.4.2. Compression of sequence number values....20
7.4.3. Decompression of sequence number values....20
7.5. Header formats....21
7.5.1. Static information packet, initialization....21
7.5.2. Context update packet....22
7.5.3. Compressed packets....24
7.5.4. Context request packet....27
8. UDP checksum....28
9. Supporting multiple packet streams....29
10. Link-layer considerations....29
11. Simulated performance results....29
11.1. Simulated scenario....29
11.2. Input data....30
11.3. Influence of pre-HC links....30
11.4. Used link layers....30
11.4.1. PPP in HDLC-like framing....31
11.4.2. Link-layer with partial checksum (LLPC)....31
11.5. The cellular link....32
11.6. Compression performance....32
11.7. Robustness results....34
11.8. CRC strength considerations....37
12. Conclusions....37
13. Intellectual property considerations....37
14. References....38

15. Authors' addresses.....39

Appendix A. Detailed classification of header fields.....40
A.1. IPv6 header fields.....40
A.2. IPv4 header fields.....40
A.3. UDP header fields.....41
A.4. RTP header fields.....41
A.5. Summary.....42

Appendix B. Mapping of ID and AD fields.....43

Document history

00	1999-06-22	First release.
01	1999-09-01	Only small corrections and modifications. The Sequence Number field was erroneously labeled Destination Port in the "Context update header".

1. Introduction

During the last five years, two communication technologies in particular have become commonly used by the general public: cellular telephony and the Internet. Cellular telephony has provided its users with a revolutionary possibility to always be reachable with reasonable service quality no matter where they are. However, up to now the main service provided has been speech. With the Internet, the conditions have been almost the opposite. While flexibility for all kinds of usage has been its strength, its focus has been on fixed connections and large terminals, and the experienced quality of some services (like Internet telephony) has generally been low.

Today, IP-telephony is gaining momentum due to improved technical solutions. It seems reasonable to believe that IP in the years to come will be a commonly used way to carry telephony. Some future cellular telephony links might also be based on IP and IP-telephony. Cellular phones may have IP stacks supporting not only audio and video, but also web browsing, email, gaming, etc.

The scenario we are envisioning might then be the one in Figure 1.1, where two mobile terminals are communicating with each other. Both are connected to base stations over cellular links and the base stations are connected to each other through a wired (or possibly wireless) network. Instead of two mobile terminals, there could of course be one mobile and one wired terminal, but the case with two cellular links is technically more demanding.

```
Mobile            Base                      Base            Mobile
Terminal          Station                   Station         Terminal


      |  ~   ~   ~  \ /                      \ /  ~   ~   ~   ~  |
      |              |                        |                  |
   +--+              |                        |               +--+
   |  |              |                        |               |  |
   |  |              |                        |               |  |
   +--+              |                        |               +--+
                     |                        |
                     |========================|

         Cellular              Wired               Cellular
         Link                  Network             Link
```

Figure 1.1 : Scenario for IP telephony over cellular links

It is obvious that the wired network can be IP-based. With the cellular links, the situation is less clear. IP could be terminated in the fixed network, and special solutions be implemented for each supported service over the cellular link. However, this would limit

the flexibility of the services supported. If technically and economically feasible, a solution with pure IP all the way from terminal to terminal would have certain advantages. However, to make IP-all-the-way a viable alternative, a number of problems have to be addressed, especially regarding bandwidth efficiency.

For cellular phone systems, it is of vital importance to use the scarce radio resources in an efficient way. A sufficient number of users per cell is crucial, otherwise deployment costs will be prohibitive [CELL]. The quality of the voice service should also be as good as in today's cellular systems. It is likely that even with support for new services, lower quality of the voice service is acceptable only if costs are significantly reduced.

A problem with IP over cellular links when used for interactive voice conversations is the large header overhead. Speech data for IP-telephony will most likely be carried by RTP [RTP]. A packet will then, in addition to link-layer framing, have an IP [IPv4] header (20 octets), a UDP [UDP] header (8 octets), and an RTP header (12 octets) for a total of 40 octets. With IPv6 [IPv6], the IP header is 40 octets for a total of 60 octets. The size of the payload depends on the speech coding and frame sizes used and may be as low as 15-20 octets.

From these numbers, the need for reducing header sizes for efficiency reasons is obvious. However, cellular links have characteristics that make header compression as defined in [IPHC,CRTP,PPPHC] perform less than well. The most important characteristic is the lossy behavior of cellular links, where a bit-error-rate (BER) as high as 1e-3 must be accepted to keep the radio resources efficiently utilized [CELL]. In severe operating situations, the BER can be as high as 1e-2. The other problematic characteristic is the long round-trip time (RTT) of the cellular link, which can be as high as 100-200 milliseconds [CELL]. A viable header compression scheme for cellular links must be able to handle loss, on the link between the compression and decompression point as well as loss before the compression point.

Bandwidth is the most costly resource in cellular links. Processing power is very cheap in comparison. Implementation or computational simplicity of a header compression scheme is therefore of less importance than its compression ratio and robustness.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

BER

Bit Error Rate. Cellular radio links have a rather high BER. In this document BER is usually given as a frequency, but one also needs to consider the error distribution as bit errors are not independent. In our simulations we use a channel with a certain BER, the error distribution is according to a realistic channel [WCDMA].

Cellular links

Wireless links where the BER is rather high in order to achieve an efficient system overall.

Compression efficiency

The performance of a header compression scheme can be described with two parameters, compression efficiency and robustness. The compression efficiency is how much header sizes are reduced by the compression scheme.

Compression profile

A compression profile is a specification of how to compress a certain kind of packet stream over a certain kind of link. Compression profiles provide the details of the header compression framework introduced in this document.

Context

The context is the state which the compressor uses to compress a header and the decompressor uses to decompress a header. The context is the uncompressed version of the last header sent (compressor) or received (decompressor) over the link, except for fields in the header that are included "as-is" in compressed headers or can be inferred from, e.g., the size of the link-level frame. The context can also contain additional information describing the packet stream, for example the typical inter-packet increase in sequence numbers or timestamps.

Context damage

When the context of the decompressor is not consistent with the context of the compressor, header decompression will fail. This situation can occur when the context of the decompressor has not

been initialized properly or when packets have been lost or damaged between compressor and decompressor. Packets which cannot be decompressed due to inconsistent contexts are said to be lost due to context damage.

Context repair mechanism

To avoid excessive context damage a context repair mechanism is needed. Context repair mechanisms can be based on explicit requests for context updates, periodic updates sent by the compressor, or methods for local repair at the decompressor side.

FER

Frame Error Rate. The FER considered in this document includes the frames lost on the channel between compressor and decompressor and frames lost due to context damage. FER is here defined to be identical to packet loss rate.

Ideal header compression scheme

An ideal header compression scheme introduced and defined for comparison purposes. It is not realistic. The ideal scheme performs like CRTP would do if used over error-free links to compress input data without irregular changes in its header fields. The compressed header of the ideal header compression scheme is always two bytes, the scheme never loses packets due to context damage, and it does not need to initialize the decompressor context.

Header compression CRC

A 10-bit CRC computed by the compressor and included in each compressed header. Its main purpose is to provide a way for the decompressor to reliably verify the correctness of reconstructed headers. What values the CRC is computed over depends on the packet type it is included in, typically it covers the original header.

Pre-HC links

Pre-HC links are all links before the header compression point. If we consider a path with cellular links as first and last hops, the Pre-HC links for the compressor at the last link are the first cellular link plus the wired links in between.

Robustness

The performance of a header compression scheme can be described with two parameters, compression efficiency and robustness. A robust scheme tolerates errors on the link over which header compression is taken place without losing additional packets, introducing additional errors, or using more bandwidth.

Spectrum efficiency

Radio resources are limited and expensive. Therefore they must be used efficiently to make the system economically feasible. In cellular systems this is achieved by maximizing the number of users served within each cell, while the quality of the provided services are at an acceptable level. A consequence of efficient spectrum use is high BER.

Timestamp delta

The timestamp delta is the increase in the timestamp value between two consecutive packets.

3. Existing header compression schemes

The original header compression scheme, CTCP [VJHC], was invented by Van Jacobson. CTCP compressed the 40 octet IP+TCP header to 4 bytes.

Header compression methods maintains a context, which is essentially the uncompressed version of the last header sent over the link, at both compressor and decompressor. Compression and decompression is done relative to the context. When compressed headers carry differences from the previous header, each compressed header will update the context of the decompressor. When a packet is lost between compressor and decompressor, the context of the decompressor will be brought out of sync since it is not updated correctly. A header compression method must have a way to repair the context, i.e., bring it in sync, after such events.

The CTCP compressor detects transport-level retransmissions and sends a header that updates the context completely when they occur. This repair mechanism does not require any explicit signaling between compressor and decompressor.

CRTP [CRTP, IPHC] by Casner and Jacobson is a header compression scheme that compresses 40 octet IPv4/UDP/RTP headers to a minimum of 2 octets when no UDP checksum is present. If the UDP checksum is present, the minimum CRTP header is 4 octets. CRTP cannot use the same repair mechanism as CTCP since UDP/RTP does not retransmit. Instead, CRTP uses explicit signaling messages from decompressor to compressor, called CONTEXT_STATE messages, to indicate that the context is out of sync. The link roundtrip time will thus limit the speed of this context repair mechanism.

On lossy links with long roundtrip times, such as most cellular links, CRTP does not perform well. Each lost packet over the link causes several subsequent packets to be lost since the context is out of sync during at least one link roundtrip time. This behavior is documented in [CRTPC]. For voice conversations such long loss events

will degrade the voice quality. Moreover, bandwidth is wasted by the large headers sent by CRTP when updating the context. [CRTPC] found that CRTP performed much worse than an ideal header compression scheme for a lossy cellular link. It is clear that CRTP alone is not a viable header compression scheme for cellular links.

To avoid losing headers due to the context being out of sync, CRTP decompressors can attempt to repair the context locally by using a mechanism known as TWICE. Each CRTP packet contains a counter which is incremented by one for each packet sent out by the CRTP compressor. If the counter increases by more than one, at least one packet was lost over the link. The decompressor then attempts to repair the context by guessing how the lost packet(s) would have updated it. The guess is then verified by decompressing the packet and checking the UDP checksum - if it succeeds, the repair is deemed successful and the packet can be forwarded or delivered. TWICE gets its name from the observation that when the compressed packet stream is regular, the correct guess is to apply the update in the current packet twice. [CRTPC] found that even CRTP with TWICE lost around two times as many packets than the ideal header compression scheme. TWICE improves CRTP performance significantly. However, there are several problems with using TWICE:

1) It becomes mandatory to use the UDP checksum:

   - the compressed header size increases by 100% to 4 octets.

   - most speech codecs developed for cellular links tolerate errors in the encoded data. Such codecs will not want to enable the UDP checksum, since they want damaged packets to be delivered.

   - errors in the payload will make the UDP checksum fail when the guess is correct (and might make it succeed when it is wrong).

2) Loss in an RTP stream that occur before the compression point will make updates in CRTP headers less regular. Simple-minded versions of TWICE will then perform badly. More sophisticated versions would need more repair attempts before finding the correct update.

4. Desired improvements

The major problem with CRTP is that it is not sufficiently robust against packets being damaged between compressor and decompressor. A viable header compression scheme must be less fragile. This increased robustness must be obtained without increasing the compressed header size; a larger header would make IP telephony over cellular links economically unattractive.

A major cause for CRTP:s bad performance over cellular links is the long link roundtrip time, during which many packets are lost when the

context is out of sync. That problem can be attacked directly by finding ways to reduce the link roundtrip time. Future generations of cellular technologies may indeed achieve lower link roundtrip times. However, they will probably always be rather high [CELL]. The benefits in terms of lower loss and smaller bandwidth demands if the context can be repaired locally will be present even if the link roundtrip time is decreased. A reliable way to detect a successful context repair is then needed.

One might argue that a better way to solve the problem is to improve the cellular link so that packet loss is less likely to occur. It would of course be nice if the links were almost error free, but such a system would not be able to support a sufficiently large number of users per cell and would thus be economically unfeasible [CELL].

One might also argue that the speech codecs should be able to deal with the kind of packet loss induced by CRTP, in particular since the speech codecs probably must be able to deal with packet loss anyway if the RTP stream crosses the Internet. While the latter is true, the kind of loss induced by CRTP is difficult to deal with. It is usually not possible to hide a loss event where well over 100 ms worth of sound is completely lost. If such loss occurs frequently at both ends of the path, the speech quality will suffer.

5. Proposed solution

We propose a solution which is heavily geared towards local repair of the context. What is needed is a reliable way to detect when a repair attempt has succeeded, plus possibly hints to the decompressor about how the header fields have changed.

The key element of our header compression scheme is that compressed headers carry a 10-bit strong checksum computed over the header before compression. This provides a reliable way to detect whether decompression and context repair has succeeded. In addition to these 10 bits, the header will contain codes and additional information as needed for decompression.

A completely general solution cannot achieve compression rates high enough to make IP telephony over cellular economically feasible. On the other hand the solution needs to be extendable so that other kinds of packet streams can also be compressed well. Therefore, we envision a scheme where the basic framework is supplemented with a set of compression profiles, where each compression profile provides the exact details on how a packet stream is to be compressed and decompressed. The use of compression profiles allows the basic framework to be adapted to the properties of packet streams as well as various link properties.

5.1. Base header format

There is only one kind of base header in this scheme. Distinguishing packet streams and packet types is necessary, but some of that information may be available from the underlying technology, for example if each packet stream has its own logical channel. To avoid wasting precious header bits, we decided to leave it to the compression profile to decide how to distinguish packet types and packet streams. This allows efficient use of header bits overall when the link-layer does not carry explicit information on packet types.

The header format is shown below. The only fields defined by the basic framework are the Header Compression CRC and Code fields. The semantics of the Code field is determined by the compression profile used. By inspecting the Code field, the decompressor knows whether the Extension field is present and its size, and also if the Payload field is present.

```
                        1         1
 0                  9 0         5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - - ~ ~ - - + - - ~ ~ - - +
| Header Compr. CRC |   Code    |  Extension  |   Payload   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - - ~ ~ - - + - - ~ ~ - - +
```

Header Compression CRC	10-bit CRC covering whatever the current compression profile specifies for the packet type.
Code	6-bit code whose semantics is defined by the compression profile. Determines whether the next two fields are present.
Extension	If present, an extended code and/or values as determined by Code and compression profile.
Payload	If present, the payload of the original packet.

A compression profile can define headers which do not have a corresponding original packet. An example would be to send non-changing fields of a packet stream as a separate packet. Another example would be to send packets to update the RTP-timestamp field even when no RTP packets arrive, in order to decrease the increment in the RTP-timestamp field when a packet does arrive. For such compressed headers, the compression profile must specify over what values the Header Compression CRC is computed.

5.2. Data structures

The compression scheme needs to maintain a context for compression and decompression of a packet stream. The context must be kept updated at both compressor and decompressor. The context is essentially the header of the last packet transmitted, and includes all static header fields plus some fields that change more or less frequently. If the compression profile used is designed to handle a certain amount of packet loss on the link, both compressor and decompressor will typically keep information about earlier packets; packets that arrived before the current packet. Finally, there may be packet stream related information such as the timestamp delta or a list of which CSRC items that have occurred in the packet stream.

5.3. Header compression profiles

The details on how a packet stream is to be compressed and decompressed is determined by a compression profile. Over a link a number of profiles can be active, but for each logical channel exactly one profile is active. How to determine what profile to use for which channel and how that is negotiated between compressor and decompressor is not specified in this document.

One way to select a profile can be to have a common channel over which a general-purpose header compression scheme, perhaps CRTP, is used. When its found that a packet stream is compressible, and the change pattern of its fields has been observed, it is switched to another channel where a suitable compression profile is used.

Profiles can be defined to compress one packet stream only, in which case the link layer must be able to separate packet streams. Profiles can also be defined for compression of more than one packet stream in which case the profile must provide a way to distinguish between the packet streams.

Important parameters to consider when designing a compression profile are:

- what support there is from the link-layer, such as the number of packet streams per channels, and if it can indicate packet types.

- the rate and pattern of loss of the channel.

- the rate and pattern of loss before the compression point.

- what kinds of packet streams to compress (IPv6, IPv4, RTP, TCP, etc).

- the pattern of change of the changing fields.

When these things have been considered, the specifics of the profile can be determined. The profile must specify:

- the exact semantics of the Code field, which will include packet types, packet formats, extensions, etc.

- what the Header Compression CRC covers for all packet types.

- the information needed in the contexts for compression and decompression, which will include history information and properties of the packet stream.

- procedures for compression and decompression.

- how compression is initiated (full headers, or by other means).

- description of context repair mechanisms.

Chapter 7 is an example of a compression profile for IP telephony over cellular links.

6. Classification of header fields

The IP/UDP/RTP header fields can be classified by the way they are expected to change. First of all, we classify them as:

INFERRED	These fields contain values that can be inferred from other values, for example the size of the frame carrying the packet, and thus must not be handled at all by the compression scheme.
STATIC	These fields are expected to be constant during the lifetime of the packet stream. Static information must in some way be communicated once.
STATIC-KNOWN	These static fields are expected to have well known values and are therefore not needed to be communicated at all.
CHANGING	These fields are expected to vary in some way, either randomly, within a limited scope, with constant values, or by other means.

All fields of the IP/UDP/RTP headers are classified in appendix A. Table 6.1 summarizes this for IP/UDP/RTP. The interval for the CHANGING fields in Table 6.1 reflect the varying size of the CSRC list of the RTP header.

Class \ IP ver	IPv6 (octets)	IPv4 (octets)
STATIC	42 +6 bits	16 +3 bits
STATIC-KNOWN	1 +6 bits	4 +1 bit
CHANGING	11.5(-71.5)	13.5(-73.5)
INFERRED	4	6
Total	60(-120)	40(-100)

Table 6.1 : size of field classes

The information carried in the STATIC fields have to be transferred once, and the profile MUST specify a way for doing this. It MUST also handle the CHANGING fields, and that SHOULD be done efficiently based on the expected change patterns. The information in INFERRED and STATIC-KNOWN fields SHOULD NOT be transmitted at all. The values of INFERRED fields can be computed from other information known to the decompressor. The values of STATIC-KNOWN fields are implicitly defined by the compression profile.

7. A header compression profile for IP-telephony packet streams

This section is an example showing how the framework outlined in 5 could be instantiated for compressing the headers of an IP telephony packet stream. Both IPv6 and IPv4 are covered, so this section actually defines two profiles: one for IPv6/UDP/RTP packet streams and one for IPv4/UDP/RTP packet streams.

7.1. Usage scenarios, environment and requirements

This profile is intended for IP-telephony over cellular links with high error rates. The scheme is designed to successfully handle loss of at least two consecutive packets over the link, without introducing any additional loss. The packet streams MUST be separated such that each packet stream has its own logical channel. The link layer MUST provide information about the size of each packet sent over the link. The scheme does not rely on there being a link-layer checksum.

As a cellular link with similar characteristics is expected at the other end of the connection, the scheme is also designed to handle two consecutive lost packet before the compression point without increasing the size of the compressed header. The profile is also designed to handle reordering of single packets before the compression point without increasing the size of compressed headers.

7.2. Analysis of change patterns of header fields

To design a suitable coding for the CHANGING header fields, their change patterns need to be analyzed. Table 7.1 summarizes the expected change patterns of these fields.

Field \ Frequency	~1/pkt	~1/talkspurt	seldom
IPv4 Identification	X	X	X
IP TOS/Tr. Class			X
IP TTL/Hop Limit			X
UDP Checksum	X		X
RTP Marker		X	
RTP Payload Type			X
RTP Seq. Number			X
RTP Timestamp		X	X
RTP CSRC		X	X
Timestamp Delta			X

Table 7.1 : Change frequencies of CHANGING header fields

An X in the first column of Table 7.1 indicates random changes in all or almost all packets. Column two is for rather infrequent changes, on the order of one per talkspurt, while the third column is for very infrequent changes. The third column is also used for changes that are constant and predictable; an example would be the RTP sequence number which will increment by one for each packet.

Table 7.1 does not consider changes caused by loss or reordering before the compression point. Such problems are dealt with by the mechanisms in section 7.4. The following subsections discusses the various header fields in detail.

7.2.1. IPv4 Identification

The Identification field (IP ID) of the IPv4 header is there to identify which fragments constitute the same datagram when reassembling fragmented datagrams. The IPv4 specification does not specify exactly how this field is to be assigned values, only that each packet should get an IP ID that is unique for the source-destination pair and protocol for the time the datagram (or any of its fragments) could be alive in the network. This means that assignment of IP ID values can be done in various ways, which we have separated into three classes.

Sequential

This assignment policy keeps a separate counter for each outgoing packet stream and thus the IP ID value will increment by one for each packet. When RTP is used on top of UDP and IP, this means that the IP ID value would follow the RTP sequence number. The change pattern corresponds to column three of Table 7.1.

Random

Some IP stacks assign IP ID values using a pseudo-random number generator. There is thus no correlation between the ID values of subsequent datagrams. Therefore there is no way to predict the IP ID value for the next datagram. For header compression purposes, this means that the IP ID field needs to be sent uncompressed with each datagram, resulting in two extra octets of header. IP stacks in cellular terminals SHOULD NOT use this IP ID assignment policy. The change pattern corresponds to column one of Table 7.1.

Sequential jump

This class originates from IP stacks that that do sequential assignment of IP ID values, and uses the same counter for all connections. When the sender is involved in more than one communication simultaneously, the IP ID can increase by more than one. The IP ID values will be much more predictable and require less bits to transfer than random values, and the packet-to-packet increment (determined by the number of active outgoing packet streams) will usually be limited. This change pattern corresponds to column two of Table 7.1.

This profile is designed to handle sequential IP ID fields. Modified profiles are required if the behavior is different. Designers of IP stacks for cellular terminals SHOULD use the Sequential policy.

7.2.2. IP Traffic-Class / Type-Of-Service

The Traffic-Class (IPv6) or Type-Of-Service (IPv4) field is expected to be constant during the lifetime of a packet stream or to change relatively seldom, corresponding to column three of Table 7.1.

7.2.3. IP Hop-Limit / Time-To-Live

The Hop-Limit (IPv6) or Time-To-Live (IPv4) field is expected to be constant during the lifetime of a packet stream or to alternate between a limited number of values, therefore it corresponds to column three of table 7.1.

7.2.4. UDP Checksum

The UDP checksum is optional. If disabled, it corresponds to column three of Table 7.1 because its value is constant (zero). If enabled, its value depends on the payload which for compression purposes will be equivalent to it changing randomly with every packet. Therefore it belongs in column one of Table 7.1.

This profile does not handle UDP checksums, i.e., it assumes that the UDP checksum is disabled. Chapter 8 discusses ways to define profiles for packet streams with enabled UDP checksums.

7.2.5. RTP Marker

The marker bit is assumed to be set only in the first packet of a talkspurt, corresponding to column two of Table 7.1.

7.2.6. RTP Payload Type

Changes of the RTP payload type within a packet stream is expected to be a rare case. Applications could adapt to congestion by changing payload type and/or frame sizes, but that is not expected to happen frequently. This patterns corresponds to column three of Table 7.1.

7.2.7. RTP Sequence Number

The RTP sequence number will be incremented with one for each packet. This corresponds to column 3 of Table 7.1.

7.2.8. RTP Timestamp

As long as there are no pauses in the audio stream, the RTP timestamp will be incremented with a constant value, corresponding to the number of samples in the (constant size) speech frame. It will thus follow the RTP sequence number. This corresponds to column three of Table 7.1. When there has been a silent period and a new talkspurt begins, the timestamp will jump in proportion to the length of the silent period. Such cases correspond to column two of Table 7.1.

7.2.9. RTP Contributing Sources (CSRC)

The participants of a session, which are identified by the CSRC fields, are expected to be almost the same on a packet-to-packet basis with relatively additions or removals. Therefore the occurrence in column two of Table 7.1. For two-part conversations where no CSRC is present the behavior corresponds to column three of Table 7.1.

7.2.10. RTP Timestamp delta

The RTP timestamp will be incremented with a constant value (the timestamp delta) for each packet received as long as the time per speech frame is constant. Changes of the timestamp delta is expected to be rare, and correspond to column three of Table 7.1.

7.3. Packet types and code field usage

The scheme makes use of four different packet types; STATIC, CONTEXT_UPDATE, COMPRESSED and CONTEXT_REQUEST. Detailed descriptions of packet formats are given in chapter 7.5.

The Code field is used to distinguish between packet types. Four code points in the Code field are reserved to identify STATIC, CONTEXT_UPDATE and CONTEXT_REQUEST packets while all other combinations are used for COMPRESSED packets. The reserved patterns are:

```
STATIC           000000
CONTEXT_UPDATE   000001
CONTEXT_REQUEST  11111-
```

For other values of the Code field, its format is as shown below.

```
+-+-+-+-+-+-+            +-+-+-+-+-+-+
|   Code    |     ->     | S-Code  |X|
+-+-+-+-+-+-+            +-+-+-+-+-+-+
```

S-Code Sequence code. 5 bits. An encoding of RTP sequence number changes for the actual and some previous packets. Its semantics is specified in 7.4.

X E(X)tension bit. X=1 indicates that an extension is present. X=0 indicates that there is no extension. Extensions are defined in section 7.5.

7.4. Robust encoding and decoding of delta information

The analysis in section 7.2 excluded changes due to loss and/or reordering before the header compression point. Such changes will have an impact on the regularity of the RTP sequence number, RTP timestamp value and, for IPv4, the IP ID value. However, as described in 7.2, both the RTP timestamp and the IP ID value are expected to follow the RTP sequence number for most packets. The task is then to communicate RTP sequence number changes in a way that tolerates two consecutive lost packets on the link between compressor and decompressor. This chapter describes how to do this by using the 5-bit S-Code.

7.4.1. Sequence number changes to handle

The source increases the RTP sequence number with one for each packet sent. However, due to losses and reordering the changes seen by the compressor will vary. If we consider our scenario in Figure 1.1 where there are cellular links at both ends of the path, it seems reasonable to set the same algorithm requirements for loss on a cellular link at the other end of the path. This implies that two consecutive packets could be lost. Loss on the wired portion of the path is assumed to be insignificant in comparison, so the maximal number of lost packets to be handled efficiently is two also before the header compression point. An example of possible sequence number changes seen by the compressor would then be as shown in Figure 7.1.

```
From sender :     1   2   3   4   5   6   7   8   9   10
Lost on path :        X   X           X       X   X
Received by HC :  1           4   5       7           10
Sequence delta :  -           3   1       2            3
```

Figure 7.1: possible sequence number changes.

Possible sequence delta value for loss are then: 1, 2 or 3

Packet reordering is uncommon, but should also be handled as long as it is single reordering (one packet arrives "early"). This requires one extra possible sequence delta, -1.

The sequence deltas to handle are thus: -1, 1, 2 and 3. When packets are duplicated in the network, the delta can be 0 (zero). We do not deal with such deltas because duplicated packets MUST NOT be sent over the cellular link; they MUST be discarded.

7.4.2. Compression of sequence number values

The sequence number encoding is based on two values called Individual Delta (ID) and Accumulated Delta (AD). These two are further encoded to one value, called the Encoded Delta (ED), which is sent in the compressed header. The values are calculated as follows.

ID: The Individual Delta is the sequence number delta from the previous packet sent from the compressor. The value can be -1, 1, 2 or 3.

AD: The Accumulated Delta is a sum of the Individual Delta values of the two previous packets sent from the compressor. The possible values of AD will be 1, 2, 3, 4, 5 and 6 (0 or -1 not possible).

ED: ID and AD are mapped to a single value, ED, as described in Appendix B. The ED value is then sent in the S-code field of each COMPRESSED packet.

The purpose of transmitting the encoded delta values is to make it possible to recover from packet loss between compressor and decompressor.

7.4.3. Decompression of RTP sequence number values

The decompressor makes attempts to decompress based on assumptions on the number of lost packets between compressor and decompressor since the last packet received and successfully decompressed. For each attempt the header is reconstructed according to the assumed number of lost packets, and the correctness is verified with the Header Compression CRC. To reconstruct the header the ID and AD values are used to calculate the sequence number corresponding to the attempt. The decompressor needs a short history of previous IDs, ADs and RTP sequence numbers to do this. The details of the algorithm is explained in the following:

Let the decompressor keep an imaginary counter, S, of the packets received and let N be the number of the current packet. S(N) is the calculated RTP sequence number for packet N. The attempts are made in the following order:

Attempt 1 - No loss : $S(N) = S(N-1) + ID(N)$

Attempt 2 - One lost : $S(N) = S(N-1) + ID(N) + AD(N) - ID(N-1)$

Attempt 3 - Two lost : $S(N) = S(N-1) + ID(N) + AD(N)$

If attempt 3 fails, more than two previous consecutive packets must have been lost between compressor and decompressor and the decompression is not guaranteed to succeed. A decompressor MAY make

additional repair attempts. A CONTEXT_REQUEST MUST be sent to the compressor to request an update to repair the context if the repair fails.

7.5. Header formats

This section defines the header formats of the four packet types together with descriptions of when and how to use them.

7.5.1. Static information packet, initialization

The STATIC packet type is a packet containing no payload but only the header fields that are expected to be constant within the lifetime of the packet stream (classified as STATIC in chapter 6). A packet of this kind MUST be sent once as the first packet from compressor to decompressor and also when requested by decompressor (see 7.5.4). The packet format is shown below for IPv6 and IPv4, respectively.

IPv6 (45 octets)

```
                                1 1             2 2             3
   0             7 8            5 6             3 4             1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Header Compr. CRC |0 0 0 0 0|0|
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |              Flow Label               |P|E| * |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                            S S R C                            |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |          Source Port          |        Destination Port       |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                                                               |
  +                                                               +
  |                                                               |
  +                        Source Address                         +
  |                                                               |
  +                                                               +
  |                                                               |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |                                                               |
  +                                                               +
  |                                                               |
  +                      Destination Address                      +
  |                                                               |
  +                                                               +
  |                                                               |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 (19 octets)

```
                                1 1             2 2             3
 0             7 8              5 6             3 4             1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header Compr. CRC |0 0 0 0 0|0|F|P|E|    *    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           S S R C                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |        Destination Port       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Source Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For this packet type, the Header Compression CRC is calculated over the entire packet, except the Header Compression CRC field, and is used at the decompressor side to verify the correctness of the packet. All other fields except the code (000000) and the unused (*) are the ordinary IP, UDP and RTP fields (F=IPv4 Don´t Fragment, P=RTP Padding, E=RTP Extension).

Only one STATIC packet is sent at each occasion. If the decompressor receives compressed headers or updates without having received a STATIC packet, the decompressor MUST request a STATIC packet.

7.5.2. Context update packet

The CONTEXT_UPDATE packet type has a header containing all changing header fields in their original, uncompressed form, and carries a payload just as ordinary COMPRESSED packets. A CONTEXT_UPDATE packet MUST be sent after the initial STATIC packet to set up the decompressor context for the first time. In addition, this update packet type MUST be used whenever the decompressor requests a context update, and when the header fields change in a way that cannot be encoded in COMPRESSED packets. The packet format is shown below for IPv6 and IPv4, respectively.

IPv6 (13 octets + CSRC List of 0-60 octets)

```
                                1 1             2 2             3
 0             7 8              5 6             3 4             1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header Compr. CRC |0 0 0 0 0|1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Traffic Class |   Hop Limit   |M| Payload Type|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Timestamp Delta    | CSRC  |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           CSRC List                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            Payload                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 (15 octets + CSRC List of 0-60 octets)

```
                                1 1             2 2             3
 0             7 8              5 6             3 4             1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Header Compr. CRC |0 0 0 0 0|1|      TOS      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Identification         |      TTL      |M| Payload Type|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Timestamp Delta    | CSRC  |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                           CSRC List                           :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                            Payload                            :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

For this packet type, the Header Compression CRC is calculated over the original packet header only. All fields except the code (000001) and the Timestamp Delta are the ordinary IP, UDP and RTP fields. The Timestamp Delta is the current delta between RTP timestamps in consecutive RTP packets.

Each time a CONTEXT_UPDATE packet is sent, the two subsequent packets MUST also be CONTEXT_UPDATE packets. This ensures that the update will succeed even when two consecutive packets are lost.

7.5.3. Compressed packets

The COMPRESSED packet type is the most commonly used one and is designed to handle ordinary changes as efficiently as possible. When changes are regular, all information is carried in the base header, with only the Header Compression CRC and the Sequence Code. The header of a COMPRESSED packet has the following format.

```
                        1
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Header Compr. CRC | S-Code  |0|
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Header Compression CRC is computed over the original packet header. Neither the S-Code nor the extension bit is included in the checksum computation.

Less regular changes of the header fields require an extension in addition to the base header. The extension is of variable size depending on the information needed to be transmitted. When there is an extension present in the COMPRESSED packet, it is indicated by the extension bit being set. The header will then have the format shown below and will include at least one extra octet of data.

```
                        1                   2
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
  | Header Compr. CRC | S-Code  |1| Type|         |
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
```

The first three bits of the extension is the Type field which specifies which kind of extension is present. This profile defines six extension types. The guiding principle for choosing extension type is to find the smallest header type that can communicate the information needed.

The extension can carry an M-field, a TS LSB field, and an Extra field. The M-field is the RTP marker bit and the TS LSB is the least significant bits of the timestamp value (the most significant bits are then expected to be unchanged since previous packets). The Extra field contains four bits indicating the presence of up to four additional header fields.

The defined extension types are shown below:

```
                 1       2
                 6 7 8 9 0 1 2 3
            - - +-+-+-+-+-+-+-+-+
 Type 0         |0 0 0|M| TS LSB|
            - - +-+-+-+-+-+-+-+-+
```

```
                 1       2                   3
                 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    Type 1      |0 0 1|M|        TS LSB         |
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                 1       2                   3
                 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    Type 2      |0 1 0|M|                TS LSB                 |
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


                 1       2
                 6 7 8 9 0 1 2 3
            - - +-+-+-+-+-+-+-+-+ - -
    Type 3      |0 1 1|M| Extra |
            - - +-+-+-+-+-+-+-+-+ - -


                 1       2                   3
                 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
    Type 4      |1 0 0|M| Extra |    TS LSB     |
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -


                 1       2                   3
                 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
    Type 5      |1 0 1|M| Extra |            TS LSB             |
            - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
```

The Extra field is a bit mask indicating which additional header fields are present. The bit mapping of the Extra field is shown below followed by a description of the various additional fields.

```
  - - +-+-+-+-+ - -          - - +-+-+-+-+ - -
      | Extra |        ->        |T H C D|
  - - +-+-+-+-+ - -          - - +-+-+-+-+ - -

                          T - Traffic Class / TOS
                          H - Hop Limit / TTL
                          C - CSRC
                          D - Timestamp Delta
```

The Traffic Class / TOS field contains the value of the original header field:

```
           8 bits
  - - +-+-+-+-+-+-+-+-+ - -
      |   TC / TOS    |
  - - +-+-+-+-+-+-+-+-+ - -
```

The Hop Limit / TTL field contains the value of the original header field:

```
           8 bits
  - - +-+-+-+-+-+-+-+-+ - -
      |   HL / TTL    |
  - - +-+-+-+-+-+-+-+-+ - -
```

The CSRC field is built up of:

- a counter of the number of CSRC items present (4 bits)
- an unused field (4 bits)
- the CSRC items, 1 to 15 (4-60 octets)

```
           1 octet    +     4 to 60 octets
  - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+~~~+-+-+-+-+-+
      | Count | Unused|      CSRC Items       |
  - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+~~~+-+-+-+-+-+
```

The Timestamp Delta field is a one octet field. We want to communicate Timestamp Delta values corresponding to 80 ms. Therefore the Timestamp Delta value communicated is not the actual number of samples, but instead the number of samples divided by 8. Thus, only Timestamp Delta values evenly divisible by 8 can be communicated in a Timestamp Delta field in an extension. On the other hand the maximum value is 255*8 = 2040 (255 ms at 8000 Hz). Delta values larger than 2040 or delta values not evenly divisible by 8 must be communicated in a CONTEXT_UPDATE packet.

```
            8 bits
  - - +-+-+-+-+-+-+-+-+ - -
      |Timestamp Delta|
  - - +-+-+-+-+-+-+-+-+ - -
```

The fields present are appended to the extension in order T-H-C-D. An example where the HL/TTL and the Timestamp Delta fields are present in a type 3 extension is shown below. When the Timestamp Delta field is present the RTP Marker will probably also be set, which is the case in this example.

```
          Type  M  Extra
     - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
         |0 1 1|1|0 1 0 1|   HL / TTL    |Timestamp Delta|
     - - +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ - -
```

When information of any kind is sent in an extension, the corresponding information MUST be sent also in two subsequent packets (either as Extensions or CONTEXT_UPDATEs) to satisfy the two-packet-loss requirement.

7.5.4. Context request packet

The CONTEXT_REQUEST packet is used by the decompressor to request a context update from the compressor. This is done when the context of the decompressor is not valid or not in sync and decompression therefore is impossible. The main reasons for an invalid context at the decompressor are:

- The context initialization failed.

- The context has been brought out of sync due to errors on the link and the context repair mechanisms have failed.

To set up the context in the first place, a STATIC packet MUST arrive to the decompressor to install the static parts of the context. Then a CONTEXT_UPDATE packet is REQUIRED to install the CHANGING parts of the header plus packet stream related information. COMPRESSED packets can be handled only after both these packets have been received successfully.

There are two kinds of CONTEXT_REQUEST packets. The first kind is used to request a new STATIC packet and is normally sent only if the first STATIC packet was lost or damaged and other packets arrive. The format of this STATIC_CONTEXT_REQUEST packet is shown below.

```
                                 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  STATIC_CONTEXT_REQUEST        | Header Compr. CRC |1 1 1 1 1|0|
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The other kind of CONTEXT_REQUEST requests a CONTEXT_UPDATE packet and is sent whenever COMPRESSED packets arrives but the decompressor fails to decompress them. The format of this UPDATE_CONTEXT_REQUEST packet is shown below.

```
                                 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  UPDATE_CONTEXT_REQUEST        | Header Compr. CRC |1 1 1 1 1|1|
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

8. UDP checksum

The profile described in section 7 does not handle packet streams with the 16-bit UDP checksum enabled. The main argument for making this assumption about IP-telephony packet streams over cellular links is that speech decoders developed for cellular links usually prefer to have damaged packets delivered rather than discarded.

If the UDP checksum is enabled its value will not be zero. The packet stream will then most likely have the UDP checksum enabled in all packets. A slightly different profile should then be used that can transfer the UDP checksum.

Applications which have enabled the UDP checksum are presumably interested in its end-to-end semantics and it is therefore important to maintain its semantics across the link. The interesting property of the UDP checksum is that it provides a way for the receiver to determine whether the packet has been damaged on its way through the network. It is not necessary to maintain the exact value of the UDP checksum as long as it reliably provides the packet-is-damaged information to the receiver.

There are at least three ways to define a compression profile that can handle the UDP checksum:

1. The UDP checksum is included as-is in all packets that carry payloads. That will obviously provide and maintain packet-is-damaged information but will increase the header size by two octets.

2. The UDP checksum can be replaced by a separate checksum which covers the payload only (this saves bits compared to case 1 only if the separate checksum is smaller than 16 bits). Together with a bit that indicates whether the UDP checksum succeeded before compression, it is easy for the decompressor to construct a UDP checksum that provides and maintains packet-is-damaged information.

3. The UDP checksum can be substituted with a single bit that indicates whether it succeeded or failed before compression. Together with information on whether a link layer checksum succeeded or not, a UDP checksum that provides and maintains packet-is-damaged information can be constructed by the decompressor.

Case 2 should be used only if there is no link layer checksum. If there is, case 3 should be used instead. Case 3 depends on there being a checksum in the link layer so that it is possible to determine if the packet has been damaged.

The UDP checksum is problematic for applications that send packets where parts of the payload are insensitive to damage. Such applications would want a transport layer checksum covering only the parts of the packet where damage cannot be tolerated, such as the headers. UDP is not flexible enough for such applications.

9. Supporting multiple packet streams

The compression profile specified in chapter 7 is intended for environments where separation of packet streams are handled on other levels. To support environments where multiple packet streams share a channel, multiple contexts are needed. Compression profiles for such environments should thus define packet formats which include context identifiers (CIDs).

10. Link-layer considerations

The base scheme requires a framing mechanism from the link layer, but nothing else. Neither packet type indication nor error detection in the form of checksums are required (or even desired).

A link layer framing protocol such as PPP in HDLC-like framing [HDLC] would then not be the best choice, since it provides too much functionality and uses precious bits to do so. In particular its error protection policy, where damaged frames are discarded, is not appropriate and would unnecessarily increase the frame-loss rate.

11. Simulated performance results

To evaluate the performance of ROCCO and the IP telephony profile, we have simulated three header compression schemes; CRTP [CRTP], ROCCO, and the Ideal header compression scheme defined in chapter 2. Sections 11.1 to 11.5 provides the parameters used in the simulations. Sections 11.6 and 11.7 show the results.

11.1. Simulated scenario

A source generates RTP packets which are sent over a wired network to an end-system where the last link is a cellular link. The RTP stream is being compressed over the last cellular link using one of the three header compression schemes. The simulated scenario is shown in Figure 11.1.

```
                              Compression
        Source                   point                End-system
                                        _____________  +-------+
                                      / back channel\ |       |
        +----+                   +---+/               \+----+  |
        |    |--------->---------| HC|-------->-------|HD  |  |
        +----+   Internet path   +---+  Cellular link  +----+  |
                   (loss)                               |       |
                                                        +-------+
```

Figure 11.1 : Simulated scenario.

11.2. Input data

The speech source generates packets with a fixed size, 244 bits, every 20 ms (12.2 kbps), corresponding to the GSM enhanced full-rate speech codec. On top of these bits, there is a 12 bit application CRC, making up a total of 256 bits (32 bytes).

The length of the talk spurts and the silent intervals between them are both exponentially distributed with an expected length of 1 second. Silence suppression is used, meaning that no data is transmitted during silent periods.

11.3. Influence of pre-HC links

A worst case scenario is considered. The packet loss rate is uniformly distributed with a probability of 1 %. First degree reordering is also uniformly distributed with a probability of 1 %. No higher reordering degree is considered. The purpose of using high error probabilities is to test the capabilities of the schemes also under tough conditions. The speech quality would suffer at such error rates.

11.4. Used link layers

Two link layers are considered in the algorithm evaluation, as in [CRTPC]. One is PPP in HDLC-like framing [HDLC], which has a 16-bit CRC covering the entire frame. This implies that all damaged frames are discarded at the link layer.

The second link layer used is an imaginary framing scheme called LLPC, Link Layer with Partial Checksum. As the name implies, the CRC does not cover the whole frame. The payload (speech data) is not covered by the CRC, which fits well with speech coders that can conceal some damage. The partial checksum will increase the number of headers seen by the decompressor and hence improve header compression performance.

11.4.1 PPP in HDLC-like framing (HDLC)

PPP typically uses HDLC-like framing [HDLC]. With a 16-bit checksum and compressed Address and Control fields the frames have the following format.

```
        1          1                        2
   +----------+----------+-------------+----------+----------+
   |   Flag   | Protocol | Information |   FCS    |   Flag   |
   | 01111110 |  8 bits  |      *      | 16 bits  | 01111110 |
   +----------+----------+-------------+----------+----------+
```

The Flag only occurs once between frames if they are sent back-to-back, so the amortized framing overhead is 4 octets per frame. The checksum (FCS) is calculated over the Protocol field and the Information field (payload), but not the Flags or the checksum itself.

Any errors anywhere in the frame will cause the FCS to fail. The frame will then be discarded.

11.4.2 Link-layer with partial checksum (LLPC)

This is an imaginary framing scheme derived from the HDLC-format in 11.4.1 by adding a one-octet Length field.

```
     1          1          1                        2
+----------+----------+----------+-------------+---------+----------+
|   Flag   |  Length  | Protocol | Information |   FCS   |   Flag   |
| 01111110 |  8 bits  |  8 bits  |      *      | 16 bits | 01111110 |
+----------+----------+----------+-------------+---------+----------+
```

The Length field indicates how many octets of the payload that are covered by the FCS. It can have values from 0 to 255. The FCS covers the Length and Protocol field plus as many octets in the beginning of the Information field as indicated by the Length field. The value of the Length field must not make the FCS extend over the FCS field. When sending a FULL_HEADER packet, the Length field would have the value 60 for IPv6 (40 for IPv4), since it should protect the IP, UDP, and RTP headers. When sending a minimal COMPRESSED_RTP or ROCCO packet, the Length field would have the value 2. The amortized framing overhead for LLPC is 5 octets per frame.

Any errors in the Flag, Length, Protocol, FCS, or the initial Length octets of the Information field will cause the FCS to fail. The frame will then be discarded. Errors in the Information field after the first Length octets will not affect the FCS and will not cause the frame to be discarded.

11.5. The cellular link

The cellular link is a WCDMA channel simulated with the fading model in [WCDMA]. The reported bit error rates, BER, are the BERs seen by the link layer and is thus the BER after channel coding.
The back channel used in our simulations never damages the CONTEXT_REQUEST messages. The RTT is set to 120 ms.

11.6. Compression performance

Figure 11.2 shows the average header size plotted against BERs for the three header compression schemes using HDLC. For BERs below 1e-4, both CRTP and ROCCO give an average header size of just above 2 octets, 2.25 for CRTP and 2.15 for ROCCO. The average header size for CRTP starts to increase when the BER gets higher than 1e-4 and at BER 1e-3 it is 3.35 octets. For ROCCO the average header size is constantly 2.15 octets up to BER 1e-3. For higher BERs it increases slightly to reach 2.75 octets at BER 1e-2, where CRTP has an average header size of 5.20 octets.

Figure 11.3 shows the same plots for LLPC. The average header size for ROCCO remains constant at 2.15 octets up to a BER of 5e-3. CRTP header size on the other hand starts to increase at BER 3e-4. The average header size for CRTP is 2.70 octets at BER 1e-3 compared to 2.15 for ROCCO. At BER 1e-2, CRTP gives an average header size of 4.20 octets, and ROCCO 2.25 octets. The difference between CRTP and ROCCO is mainly that the latter tolerates losing up to 2 packets before it needs a context update packet, while CRTP needs a context update for each loss. ROCCO therefore requires less updates than CRTP introducing less header overhead and loosing a significantly lower number of packets.

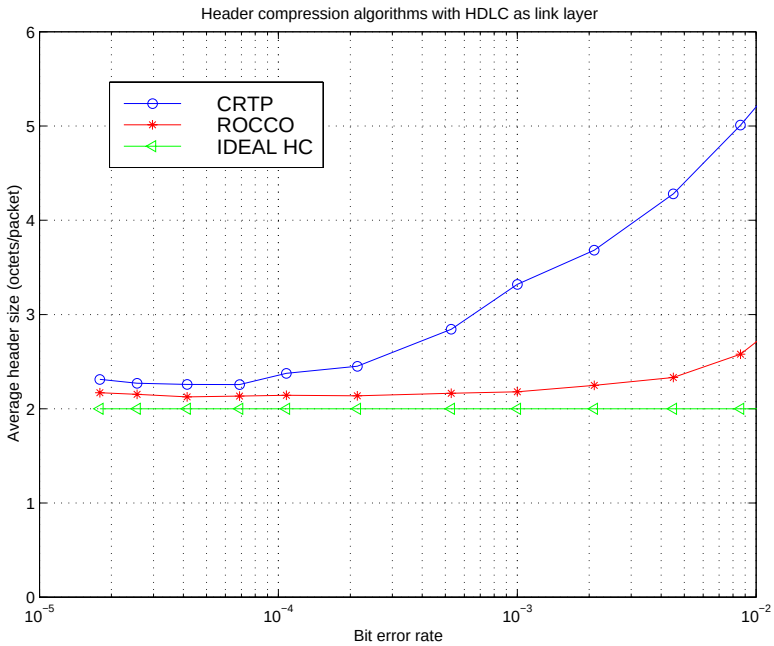

Figure 11.2 : Average header sizes with HDLC framing

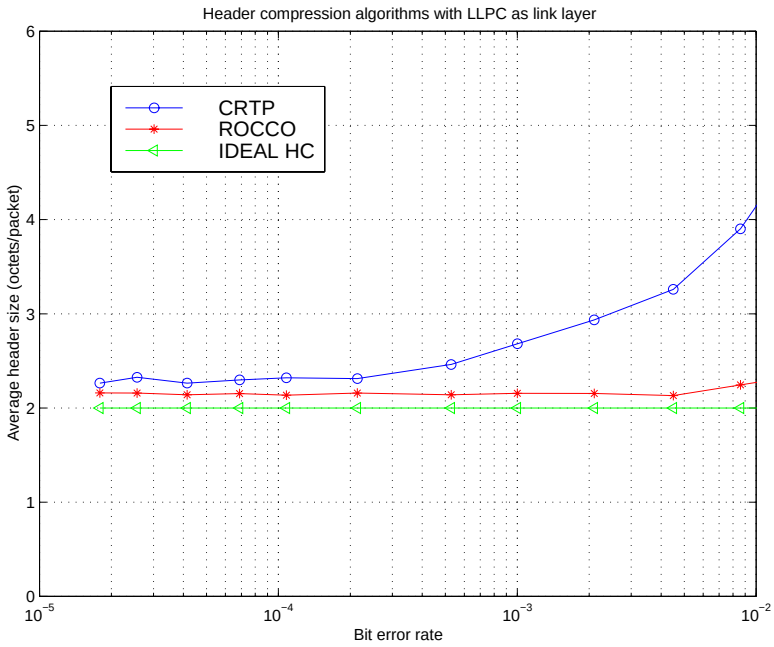

Figure 11.3 : Average header sizes with LLPC framing

11.7. Robustness results

A packet is considered as lost if it is not passed up to the application (speech codec), meaning that a packet with errors in the payload for LLPC is not regarded as lost as long as it is deemed ok by the header compression scheme.

In Figure 11.4 the FER for HDLC is shown for the three header compression schemes. At BER 1e-4 we have for CRTP 0.78 % FER, for ROCCO 0.18 % and ideal HC gives 0.14 % FER. When increasing the BER to 1e-3, CRTP gives 16.56 % FER, ROCCO 3.69 % and ideal HC 3.36 %.

Figure 11.5 shows the FER for LLPC. ROCCO and ideal HC have a FER of 0.98 % and 0.69 %, respectively, while CRTP has a FER of 5.27 %. Given the performance of the ideal scheme, it is clear that most of the CRTP loss is due to context damage. ROCCO is close to the ideal scheme until the BER gets so high that more than 2 consecutive packets often are lost on the link.

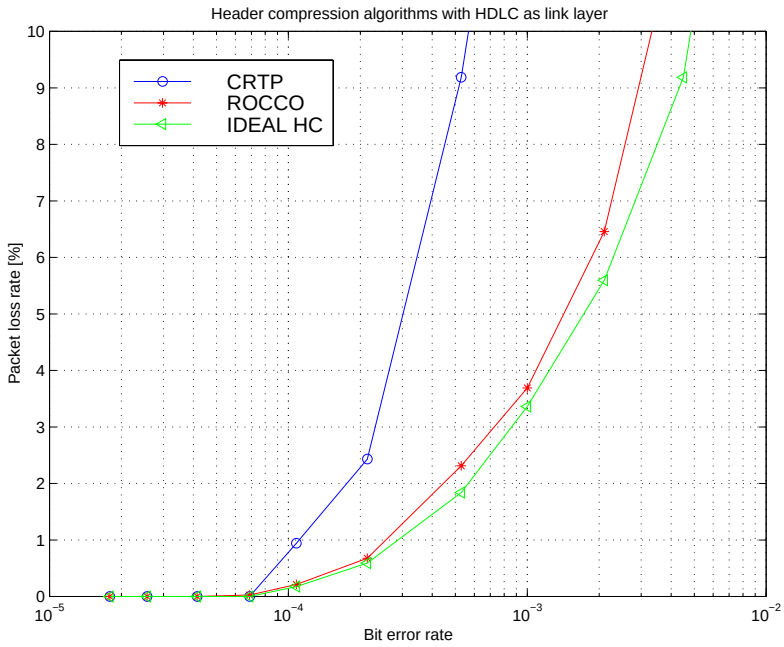

Figure 11.4 : Packet loss rate versus BER with HDLC framing

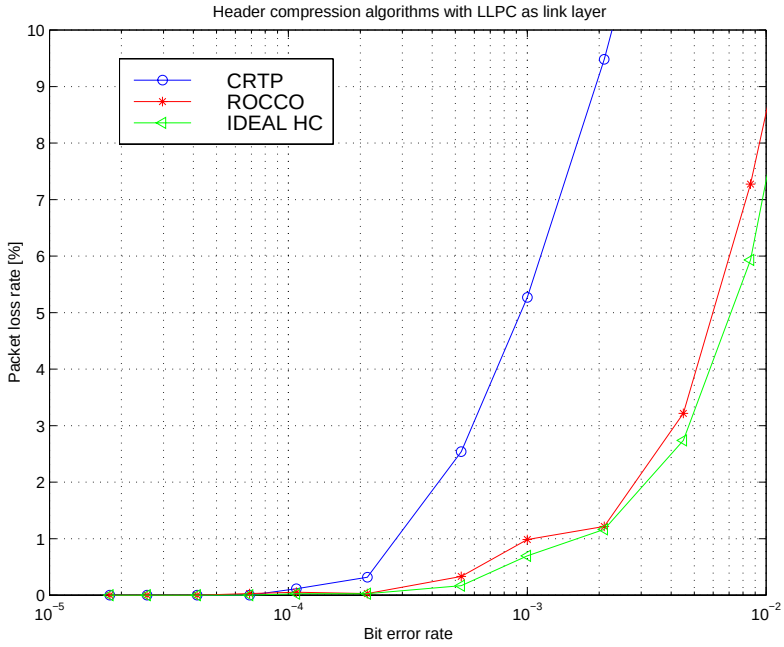

Figure 11.5 : Packet loss rate versus BER with LLPC framing

Figures 11.6 and 11.7 show the loss pattern for CRTP and ROCCO with HDLC and LLPC at BER 1e-3. It is evident from these figures that the majority of loss events with CRTP are such that around 7 consecutive frames are lost. The link roundtrip time in these simulation were 120 ms and the packet rate 50 packets per second, which means that a single discarded frame causes 6 additional frames to be lost due to context damage. For ROCCO, almost all loss events include 1 or 2 lost frames, which means that it does not suffer from context damage.

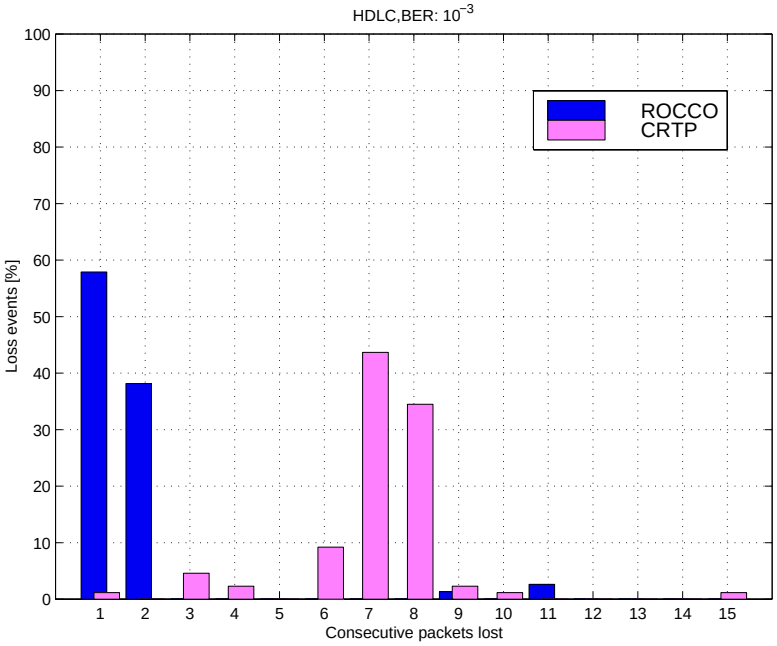

Figure 11.6 : Packet loss patterns for CRTP and ROCCO with HDLC

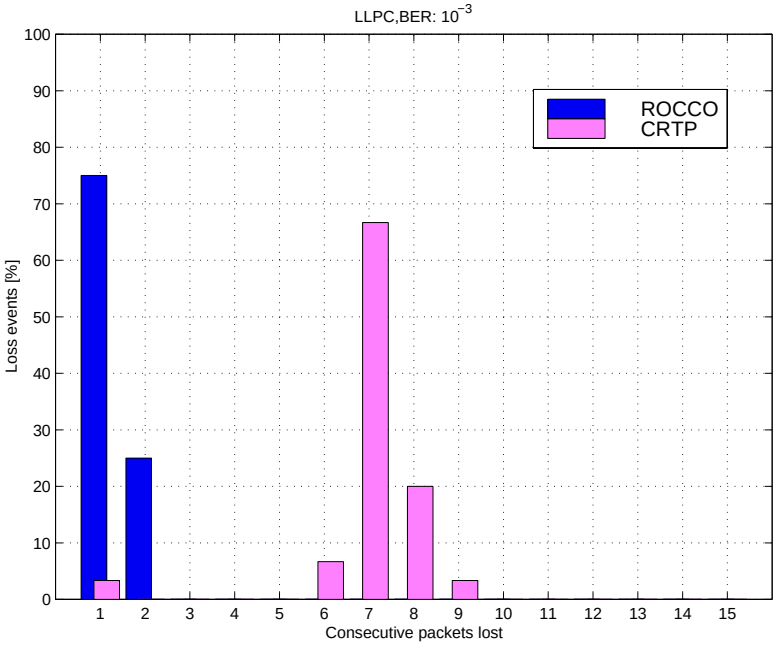

Figure 11.7 : Packet loss patterns for CRTP and ROCCO with LLPC

11.8. CRC strength considerations

The 10 bits of CRC is used to verify guesses when reconstructing the header. The only header fields with bits changing between guesses are the IP identification, RTP sequence number and RTP timestamp. More than 300,000 different combinations of these fields have gone through a CRC calculation without letting any erroneous packets through. That represents about 7 minutes of speech activity and argues therefore that 10 bits of CRC is enough to verify the correctness of the guessed header.

12. Conclusions

This document specifies a framework for header compression over lossy links based on checksums and local repairs: ROCCO. It also defines compression profiles suited for compression of RTP/UDP/IP headers in IP telephony packet streams.

The compression profile is evaluated by simulations over cellular links with high but not unrealistic error rates. Its performance is excellent and very close to the performance of an ideal header compression scheme. Compared to CRTP, the packet loss rate of ROCCO is around 4-6 times less over links with high error rates. Moreover, it does not, as does CRTP, introduce loss events involving many packets.

CRTP on the other hand performs well when error rates are low. It is also general in the sense that it is not very dependent on the properties of the RTP streams it compresses. CRTP is a good candidate for channels with low error rates and in cases when many different RTP streams are intermixed.

13. Intellectual property considerations

Ericsson has filed patent applications that might possibly have technical relations to this contribution. For an Ericsson statement regarding these applications and possible patents affecting this proposal, see the IETF IPR statements page:
http://www.ietf.org/ipr.html

14. References

[UDP] Jon Postel, "User Datagram Protocol", RFC 761, August 1980.

[IPv4] Jon Postel, "Internet Protocol", RFC 791, September 1981.

[IPv6] Steven Deering, Robert Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RTP] Henning Schulzrinne, Stephen Casner, Ron Frederick, Van Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[HDLC] William Simpson, "PPP in HDLC-like framing", RFC 1662, July 1994.

[VJHC] Van Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC 1144, February 1990.

[IPHC] Mikael Degermark, Bjorn Nordgren, Stephen Pink, "IP Header Compression", RFC 2507, February 1999.

[CRTP] Steven Casner, Van Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, February 1999.

[PPPHC] Mathias Engan, Steven Casner, Carsten Bormann, "IP Header Compression over PPP", RFC 2509, February 1999.

[CRTPC] Mikael Degermark, Hans Hannu, Lars-Erik Jonsson, Krister Svanbro, "CRTP over cellular radio links", Internet Draft (work in progress), June 1999. <draft-degermark-crtp-cellular-00.txt>

[CELL] Lars Westberg, Morgan Lindqvist, "Realtime traffic over cellular access networks", Internet Draft (work in progress), June 1999. <draft-westberg-realtime-cellular-00.txt>

[WCDMA] "Procedure for Evaluation of Transmission Technologies for FPLMTS", ITU-R TG8-1,8-1/TEMP/233-E, September 1995.

15. Authors' addresses

Lars-Erik Jonsson	Tel: +46 920 20 21 07
Ericsson Erisoft AB	Fax: +46 920 20 20 99
Box 920	Mobile: +46 70 365 20 58
SE-971 28 Lulea, Sweden	EMail: lars-erik.jonsson@ericsson.com
Mikael Degermark	Tel: +46 920 911 88
Dept of CS & EE	Fax: +46 920 728 01
Lulea University of Technology	Moblie: +46 70 833 89 33
SE-971 87 Lulea	EMail: micke@sm.luth.se
Hans Hannu	Tel: +46 920 20 21 84
Ericsson Erisoft AB	Fax: +46 920 20 20 99
Box 920	Mobile: +46 70 378 04 73
SE-971 28 Lulea, Sweden	EMail: hans.hannu@ericsson.com
Krister Svanbro	Tel: +46 920 20 20 77
Ericsson Erisoft AB	Fax: +46 920 20 20 99
Box 920	Mobile: +46 70 531 25 08
SE-971 28 Lulea, Sweden	EMail: krister.svanbro@lu.erisoft.se

Appendix A. Detailed classification of header fields (According to chapter 7)

A.1. IPv6 header fields

Field	Size (bits)	Class
Version	4	STATIC-KNOWN
Traffic Class	8	CHANGING
Flow Label	20	STATIC
Payload Length	16	INFERRED
Next Header	8	STATIC-KNOWN
Hop Limit	8	CHANGING
Source Address	128	STATIC
Destination Address	128	STATIC

Summarizing the bits corresponding to the classes gives:

Class	Size (octets)
STATIC	34.5
STATIC-KNOWN	1.5
CHANGING	2
INFERRED	2

A.2. IPv4 header fields

Field	Size (bits)	Class
Version	4	STATIC-KNOWN
Header Length	4	STATIC-KNOWN
Type Of Service	8	CHANGING
Packet Length	16	INFERRED
Identification	16	CHANGING
Reserved flag	1	STATIC-KNOWN
Don’t fragment flag	1	STATIC
More fragments flag	1	STATIC-KNOWN
Fragment Offset	13	STATIC-KNOWN
Time To Live	8	CHANGING
Protocol	8	STATIC-KNOWN
Header Checksum	16	INFERRED
Source Address	32	STATIC
Destination Address	32	STATIC

Summarizing the bits corresponding to the classes gives:

Class	Size (octets)
STATIC	8 +1 bit
STATIC-KNOWN	3 +7 bits
CHANGING	4
INFERRED	4

A.3. UDP header fields

Field	Size (bits)	Class
Source Port	16	STATIC
Destination Port	16	STATIC
Length	16	INFERRED
Checksum	16	CHANGING

Summarizing the bits corresponding to the classes gives:

Class	Size (octets)
STATIC	4
CHANGING	2
INFERRED	2

A.4. RTP header fields

Field	Size (bits)	Class
Version	2	STATIC-KNOWN
Padding	1	STATIC
Extension	1	STATIC
CSRC Counter	4	CHANGING
Marker	1	CHANGING
Payload Type	7	CHANGING
Sequence Number	16	CHANGING
Timestamp	32	CHANGING
SSRC	32	STATIC
CSRC	0(-480)	CHANGING

Summarizing the bits corresponding to the classes gives:

Class	Size (octets)
STATIC	4 + 2 bits
STATIC-KNOWN	2 bits
CHANGING	7.5(-67.5)

A.5. Summary

If we summarize this for IP/UDP/RTP we get:

Class \ IP ver	IPv6 (octets)	IPv4 (octets)
STATIC	42 +6 bits	16 +3 bits
STATIC-KNOWN	1 +6 bits	4 +1 bit
CHANGING	11.5(-71.5)	13.5(-73.5)
INFERRED	4	6
Total	60(-120)	40(-100)

Appendix B. Mapping of ID and AD fields (According to chapter 7.4.2)

The 5 bit Sequence field is defined for all bit patterns except the ones below which are reserved.

```
00000
11111
```

This leaves 32-2=30 code points to use for an encoding of ID/AD called ED. The number of possible ID and AD values are 4 and 6 respectively. To encode all combinations, 4x6=24 code points is needed and they will therefore fit into the field. The encoding is carried out as follows:

AD	ID	ED
1	-1	001 00
2	-1	010 00
3	-1	011 00
4	-1	100 00
5	-1	101 00
6	-1	110 00
1	1	001 01
2	1	010 01
3	1	011 01
4	1	100 01
5	1	101 01
6	1	110 01
1	2	001 10
2	2	010 10
3	2	011 10
4	2	100 10
5	2	101 10
6	2	110 10
1	3	001 11
2	3	010 11
3	3	011 11
4	3	100 11
5	3	101 11
6	3	110 11

The first three bits encode the AD field and the last two the ID field. Six combinations are not used at all:

```
00001        11100
00010        11101
00011        11110
```

This Internet-Draft expires in March 2000.