                   RObust Header Compression (ROHC):
         Framework and four profiles: RTP, UDP, ESP, and uncompressed

Status of this Memo

Copyright Notice

Abstract

   This document specifies a highly robust and efficient header
   compression scheme for RTP/UDP/IP (Real-Time Transport Protocol, User
   Datagram Protocol, Internet Protocol), UDP/IP, and ESP/IP
   (Encapsulating Security Payload) headers.

   Existing header compression schemes do not work well when used over
   links with significant error rates and long round-trip times.  For
   many bandwidth limited links where header compression is essential,
   such characteristics are common.

This is done in a framework designed to be extensible.  For example,
a scheme for compressing TCP/IP headers will be simple to add, and is
in development.  Headers specific to Mobile IPv4 are not subject to
special treatment, but are expected to be compressed sufficiently
well by the provided methods for compression of sequences of
extension headers and tunneling headers.  For the most part, the same
will apply to work in progress on Mobile IPv6, but future work might
be required to handle some extension headers, when a standards track
Mobile IPv6 has been completed.

Table of Contents

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 the revolutionary possibility of always being reachable
   with reasonable service quality no matter where they are.  The main
   service provided by the dedicated terminals has been speech.  The
   Internet, on the other hand, has from the beginning been designed for
   multiple services and its flexibility for all kinds of usage has been
   one of its strengths.  Internet terminals have usually been general-
   purpose and have been attached over fixed connections.  The
   experienced quality of some services (such as Internet telephony) has
   sometimes been low.

   Today, IP telephony is gaining momentum thanks to improved technical
   solutions.  It seems reasonable to believe that in the years to come,
   IP will become 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 become more general-purpose, and may have IP
   stacks supporting not only audio and video, but also web browsing,
   email, gaming, etc.

One of the scenarios 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
```
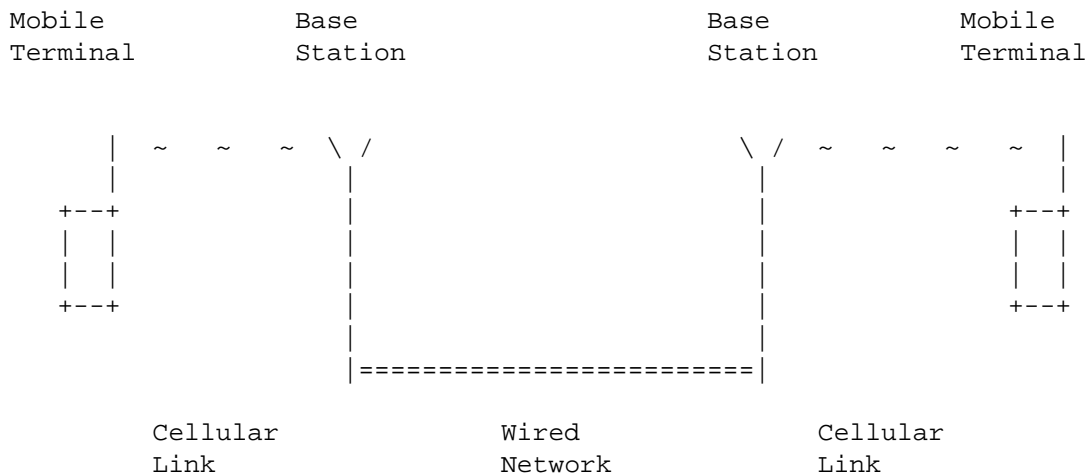
        Figure 1.1 : Scenario for IP telephony over cellular links

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 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
this a viable alternative, a number of problems have to be addressed,
in particular problems 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.  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 being 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] 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.  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.  An
additional problem is that the residual BER is nontrivial, i.e.,
lower layers can sometimes deliver frames containing undetected
errors.  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 can have a fairly high BER.
   In this document BER is usually given as a probability, but one
   also needs to consider the error distribution as bit errors are
   not independent.

   Cellular links

      Wireless links between mobile terminals and base stations.

   Compression efficiency

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

   Compression transparency

      The performance of a header compression scheme can be described
      with three parameters: compression efficiency, robustness, and
      compression transparency.  The compression transparency is a
      measure of the extent to which the scheme ensures that the
      decompressed headers are semantically identical to the original
      headers.  If all decompressed headers are semantically identical
      to the corresponding original headers, the transparency is 100
      percent.  Compression transparency is high when damage propagation
      is low.

   Context

      The context of the compressor is the state it uses to compress a
      header.  The context of the decompressor is the state it uses to
      decompress a header.  Either of these or the two in combination
      are usually referred to as "context", when it is clear which is
      intended.  The context contains relevant information from previous
      headers in the packet stream, such as static fields and possible
      reference values for compression and decompression.  Moreover,
      additional information describing the packet stream is also part
      of the context, for example information about how the IP
      Identifier field changes and 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, decompression may fail to reproduce the
      original header.  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.  Packets that are
decompressed but contain errors due to inconsistent contexts are
said to be damaged due to context damage.

Context repair mechanism

Context repair mechanisms are mechanisms that bring the contexts
in sync when they were not.  This is needed to avoid excessive
loss due to context damage.  Examples are the context request
mechanism of CRTP, the NACK mechanisms of O- and R-mode, and the
periodic refreshes of U-mode.

Note that there are also mechanisms that prevent (some) context
inconsistencies from occurring, for example the ACK-based updates
of the context in R-mode, the repetitions after change in U- and
O-mode, and the CRCs which protect context updating information.

CRC-DYNAMIC

Opposite of CRC-STATIC.

CRC-STATIC

A CRC over the original header is the primary mechanism used by
ROHC to detect incorrect decompression.  In order to decrease
computational complexity, the fields of the header are
conceptually rearranged when the CRC is computed, so that it is
first computed over octets which are static (called CRC-STATIC in
this document) and then over octets whose values are expected to
change between packets (CRC-DYNAMIC).  In this manner, the
intermediate result of the CRC computation, after it has covered
the CRC-STATIC fields, can be reused for several packets.  The
restarted CRC computation only covers the CRC-DYNAMIC octets.  See
section 5.9.

Damage propagation

Delivery of incorrect decompressed headers, due to errors in
(i.e., loss of or damage to) previous header(s) or feedback.

Loss propagation

Loss of headers, due to errors in (i.e., loss of or damage to)
previous header(s)or feedback.

Error detection

   Detection of errors.  If error detection is not perfect, there
   will be residual errors.

Error propagation

   Damage propagation or loss propagation.

Header compression profile

   A header compression profile is a specification of how to compress
   the headers of 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.  The profile
   concept makes use of profile identifiers to separate different
   profiles which are used when setting up the compression scheme.
   All variations and parameters of the header compression scheme
   that are not part of the context state are handled by different
   profile identifiers.

Packet

   Generally, a unit of transmission and reception (protocol data
   unit).  Specifically, when contrasted with "frame", the packet
   compressed and then decompressed by ROHC.  Also called
   "uncompressed packet".

Packet Stream

   A sequence of packets where the field values and change patterns
   of field values are such that the headers can be compressed using
   the same context.

Pre-HC links

   The Pre-HC links are all links that a packet has traversed 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.

Residual error

   Error introduced during transmission and not detected by lower-
   layer error detection schemes.

Robustness

   The performance of a header compression scheme can be described
   with three parameters: compression efficiency, robustness, and
   compression transparency.  A robust scheme tolerates loss and
   residual errors on the link over which header compression takes
   place without losing additional packets or introducing additional
   errors in decompressed headers.

RTT

   The RTT (round-trip time) is the time elapsing from the moment the
   compressor sends a packet until it receives feedback related to
   that packet (when such feedback is sent).

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 is kept at an acceptable level.  A consequence of
   efficient spectrum use is a high rate of errors (frame loss and
   residual bit errors), even after channel coding with error
   correction.

String

   A sequence of headers in which the values of all fields being
   compressed change according to a pattern which is fixed with
   respect to a sequence number.  Each header in a string can be
   compressed by representing it with a ROHC header which essentially
   only carries an encoded sequence number.  Fields not being
   compressed (e.g., random IP-ID, UDP Checksum) are irrelevant to
   this definition.

Timestamp stride

   The timestamp stride (TS_STRIDE) is the expected increase in the
   timestamp value between two RTP packets with consecutive sequence
   numbers.

2.1.  Acronyms

   This section lists most acronyms used for reference.

   AH      Authentication Header.
   CID     Context Identifier.
   CRC     Cyclic Redundancy Check.  Error detection mechanism.
   CRTP    Compressed RTP.  RFC 2508.
   CTCP    Compressed TCP.  Also called VJ header compression.  RFC 1144.
   ESP     Encapsulating Security Payload.
   FC      Full Context state (decompressor).
   FO      First Order state (compressor).
   GRE     Generic Routing Encapsulation.  RFC 2784, RFC 2890.
   HC      Header Compression.
   IPHC    IP Header Compression.  RFC 2507.
   IPX     Flag in Extension 2.
   IR      Initiation and Refresh state (compressor).  Also IR packet.
   IR-DYN  IR-DYN packet.
   LSB     Least Significant Bits.
   MRRU    Maximum Reconstructed Reception Unit.
   MTU     Maximum Transmission Unit.
   MSB     Most Significant Bits.
   NBO     Flag indicating whether the IP-ID is in Network Byte Order.
   NC      No Context state (decompressor).
   O-mode  Bidirectional Optimistic mode.
   PPP     Point-to-Point Protocol.
   R-mode  Bidirectional Reliable mode.
   RND     Flag indicating whether the IP-ID behaves randomly.
   ROHC    RObust Header Compression.
   RTCP    Real-Time Control Protocol.  See RTP.
   RTP     Real-Time Protocol.  RFC 1889.
   RTT     Round Trip Time (see section 2).
   SC      Static Context state (decompressor).
   SN      (compressed) Sequence Number.  Usually RTP Sequence Number.
   SO      Second Order state (compressor).
   SPI     Security Parameters Index.
   SSRC    Sending source.  Field in RTP header.
   CSRC    Contributing source.  Optional list of CSRCs in RTP header.
   TC      Traffic Class.  Octet in IPv6 header.  See also TOS.
   TOS     Type Of Service.  Octet in IPv4 header.  See also TC.
   TS      (compressed) RTP Timestamp.
   U-mode  Unidirectional mode.
   W-LSB   Window based LSB encoding.  See section 4.5.2.

3.  Background

   This chapter provides a background to the subject of header
   compression.  The fundamental ideas are described together with
   existing header compression schemes.  Their drawbacks and
   requirements are then discussed, providing motivation for new header
   compression solutions.

3.1.  Header compression fundamentals

   The main reason why header compression can be done at all is the fact
   that there is significant redundancy between header fields, both
   within the same packet header but in particular between consecutive
   packets belonging to the same packet stream.  By sending static field
   information only initially and utilizing dependencies and
   predictability for other fields, the header size can be significantly
   reduced for most packets.

   Relevant information from past packets is maintained in a context.
   The context information is used to compress (decompress) subsequent
   packets.  The compressor and decompressor update their contexts upon
   certain events.  Impairment events may lead to inconsistencies
   between the contexts of the compressor and decompressor, which in
   turn may cause incorrect decompression.  A robust header compression
   scheme needs mechanisms for avoiding context inconsistencies and also
   needs mechanisms for making the contexts consistent when they were
   not.

3.2.  Existing header compression schemes

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

   A general IP header compression scheme, IP header compression [IPHC],
   improves somewhat on CTCP and can compress arbitrary IP, TCP, and UDP
   headers.  When compressing non-TCP headers, IPHC does not use delta
   encoding and is robust.  When compressing TCP, the repair mechanism
   of CTCP is augmented with a link-level nacking scheme which speeds up
   the repair.  IPHC does not compress RTP headers.

   CRTP [CRTP, IPHC] by Casner and Jacobson is a header compression
   scheme that compresses 40 octets of IPv4/UDP/RTP headers to a minimum
   of 2 octets when the UDP Checksum is not enabled.  If the UDP
   Checksum is enabled, 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 round-trip time
will thus limit the speed of this context repair mechanism.

On lossy links with long round-trip 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 round-trip 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 did not perform well enough for a lossy cellular link.  It
is clear that CRTP alone is not a viable header compression scheme
for IP telephony over cellular links.

To avoid losing packets 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
derives 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 with TWICE, CRTP
doubled the number of lost packets.  TWICE improves CRTP performance
significantly.  However, there are several problems with using TWICE:

1) It becomes mandatory to use the UDP Checksum:

    - the minimal 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 do 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 the guess is
      wrong).

   2) Loss in an RTP stream that occurs 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 to succeed.

3.3.  Requirements on a new header compression scheme

   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 of the bad performance of CRTP over cellular links is
   the long link round-trip time, during which many packets are lost
   when the context is out of sync.  This problem can be attacked
   directly by finding ways to reduce the link round-trip time.  Future
   generations of cellular technologies may indeed achieve lower link
   round-trip times.  However, these will probably always be fairly
   high.  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 round-trip 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.  Such
   modifications do not appear to come for free, however.  If links were
   made (almost) error free, the system might not be able to support a
   sufficiently large number of users per cell and might thus be
   economically infeasible.

   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 completely 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 end-to-end path, the speech quality
   will suffer.

   A detailed description of the requirements specified for ROHC may be
   found in [REQ].

3.4.  Classification of header fields

   As mentioned earlier, header compression is possible due to the fact
   that there is much redundancy between header field values within
   packets, but especially between consecutive packets.  To utilize
   these properties for header compression, it is important to
   understand the change patterns of the various header fields.

   All header fields have been classified in detail in appendix A.  The
   fields are first classified at a high level and then some of them are
   studied more in detail.  Finally, the appendix concludes with
   recommendations on how the various fields should be handled by header
   compression algorithms.  The main conclusion that can be drawn is
   that most of the header fields can easily be compressed away since
   they never or seldom change.  Only 5 fields, with a combined size of
   about 10 octets, need more sophisticated mechanisms.  These fields
   are:

    - IPv4 Identification (16 bits)   - IP-ID
    - UDP Checksum (16 bits)
    - RTP Marker (1 bit)              - M-bit
    - RTP Sequence Number (16 bits)   - SN
    - RTP Timestamp (32 bits)         - TS

   The analysis in Appendix A reveals that the values of the TS and IP-
   ID fields can usually be predicted from the RTP Sequence Number,
   which increments by one for each packet emitted by an RTP source.
   The M-bit is also usually the same, but needs to be communicated
   explicitly occasionally.  The UDP Checksum should not be predicted
   and is sent as-is when enabled.

   The way ROHC RTP compression operates, then, is to first establish
   functions from SN to the other fields, and then reliably communicate
   the SN.  Whenever a function from SN to another field changes, i.e.,
   the existing function gives a result which is different from the
   field in the header to be compressed, additional information is sent
   to update the parameters of that function.

   Headers specific to Mobile IP (for IPv4 or IPv6) do not receive any
   special treatment in this document.  They are compressible, however,
   and it is expected that the compression efficiency for Mobile IP
   headers will be good enough due to the handling of extension header
   lists and tunneling headers.  It would be relatively painless to
   introduce a new ROHC profile with special treatment for Mobile IPv6
   specific headers should the completed work on the Mobile IPv6
   protocols (work in progress in the IETF) make that necessary.

4.  Header compression framework

4.1.  Operating assumptions

   Cellular links, which are a primary target for ROHC, have a number of
   characteristics that are described briefly here.  ROHC requires
   functionality from lower layers that is outlined here and more
   thoroughly described in the lower layer guidelines document [LLG].

   Channels

      ROHC header-compressed packets flow on channels.  Unlike many
      fixed links, some cellular radio links can have several channels
      connecting the same pair of nodes.  Each channel can have
      different characteristics in terms of error rate, bandwidth, etc.

   Context identifiers

      On some channels, the ability to transport multiple packet streams
      is required.  It can also be feasible to have channels dedicated
      to individual packet streams.  Therefore, ROHC uses a distinct
      context identifier space per channel and can eliminate context
      identifiers completely for one of the streams when few streams
      share a channel.

   Packet type indication

      Packet type indication is done in the header compression scheme
      itself.  Unless the link already has a way of indicating packet
      types which can be used, such as PPP, this provides smaller
      compressed headers overall.  It may also be less difficult to
      allocate a single packet type, rather than many, in order to run
      ROHC over links such as PPP.

   Reordering

      The channel between compressor and decompressor is required to
      maintain packet ordering, i.e., the decompressor must receive
      packets in the same order as the compressor sent them.
      (Reordering before the compression point, however, is dealt with,
      i.e., there is no assumption that the compressor will only receive
      packets in sequence.)

   Duplication

      The channel between compressor and decompressor is required to not
      duplicate packets.  (Duplication before the compression point,
      however, is dealt with, i.e., there is no assumption that the
      compressor will receive only one copy of each packet.)

   Packet length

      ROHC is designed under the assumption that lower layers indicate
      the length of a compressed packet.  ROHC packets do not contain
      length information for the payload.

   Framing

      The link layer must provide framing that makes it possible to
      distinguish frame boundaries and individual frames.

   Error detection/protection

      The ROHC scheme has been designed to cope with residual errors in
      the headers delivered to the decompressor.  CRCs and sanity checks
      are used to prevent or reduce damage propagation.  However, it is
      RECOMMENDED that lower layers deploy error detection for ROHC
      headers and do not deliver ROHC headers with high residual error
      rates.

      Without giving a hard limit on the residual error rate acceptable
      to ROHC, it is noted that for a residual bit error rate of at most
      1E-5, the ROHC scheme has been designed not to increase the number
      of damaged headers, i.e., the number of damaged headers due to
      damage propagation is designed to be less than the number of
      damaged headers caught by the ROHC error detection scheme.

   Negotiation

      In addition to the packet handling mechanisms above, the link
      layer MUST provide a way to negotiate header compression
      parameters, see also section 5.1.1.  (For unidirectional links,
      this negotiation may be performed out-of-band or even a priori.)

4.2.  Dynamicity

   The ROHC protocol achieves its compression gain by establishing state
   information at both ends of the link, i.e., at the compressor and at
   the decompressor.  Different parts of the state are established at
   different times and with different frequency; hence, it can be said
   that some of the state information is more dynamic than the rest.

Some state information is established at the time a channel is
established; ROHC assumes the existence of an out-of-band negotiation
protocol (such as PPP), or predefined channel state (most useful for
unidirectional links).  In both cases, we speak of "negotiated
channel state".  ROHC does not assume that this state can change
dynamically during the channel lifetime (and does not explicitly
support such changes, although some changes may be innocuous from a
protocol point of view).  An example of negotiated channel state is
the highest context ID number to be used by the compressor (MAX_CID).

Other state information is associated with the individual packet
streams in the channel; this state is said to be part of the context.
Using context identifiers (CIDs), multiple packet streams with
different contexts can share a channel.  The negotiated channel state
indicates the highest context identifier to be used, as well as the
selection of one of two ways to indicate the CID in the compressed
header.

It is up to the compressor to decide which packets to associate with
a context (or, equivalently, which packets constitute a single
stream); however, ROHC is efficient only when all packets of a stream
share certain properties, such as having the same values for fields
that are described as "static" in this document (e.g., the IP
addresses, port numbers, and RTP parameters such as the payload
type).  The efficiency of ROHC RTP also depends on the compressor
seeing most RTP Sequence Numbers.

Streams need not share all characteristics important for compression.
ROHC has a notion of compression profiles: a compression profile
denotes a predefined set of such characteristics.  To provide
extensibility, the negotiated channel state includes the set of
profiles acceptable to the decompressor.  The context state includes
the profile currently in use for the context.

Other elements of the context state may include the current values of
all header fields (from these one can deduce whether an IPv4 header
is present in the header chain, and whether UDP Checksums are
enabled), as well as additional compression context that is not part
of an uncompressed header, e.g., TS_STRIDE, IP-ID characteristics
(incrementing as a 16-bit value in network byte order? random?), a
number of old reference headers, and the compressor/decompressor
state machines (see next section).

This document actually defines four ROHC profiles: One uncompressed
profile, the main ROHC RTP compression profile, and two variants of
this profile for compression of packets with header chains that end

   in UDP and ESP, respectively, but where RTP compression is not
   applicable.  The descriptive text in the rest of this section is
   referring to the main ROHC RTP compression profile.

4.3.  Compression and decompression states

   Header compression with ROHC can be characterized as an interaction
   between two state machines, one compressor machine and one
   decompressor machine, each instantiated once per context.  The
   compressor and the decompressor have three states each, which in many
   ways are related to each other even if the meaning of the states are
   slightly different for the two parties.  Both machines start in the
   lowest compression state and transit gradually to higher states.

   Transitions need not be synchronized between the two machines.  In
   normal operation it is only the compressor that temporarily transits
   back to lower states.  The decompressor will transit back only when
   context damage is detected.

   Subsequent sections present an overview of the state machines and
   their corresponding states, respectively, starting with the
   compressor.

4.3.1.  Compressor states

   For ROHC compression, the three compressor states are the
   Initialization and Refresh (IR), First Order (FO), and Second Order
   (SO) states.  The compressor starts in the lowest compression state
   (IR) and transits gradually to higher compression states.  The
   compressor will always operate in the highest possible compression
   state, under the constraint that the compressor is sufficiently
   confident that the decompressor has the information necessary to
   decompress a header compressed according to that state.

```
   +----------+                   +----------+                   +----------+
   | IR State |   <-------->      | FO State |   <-------->      | SO State |
   +----------+                   +----------+                   +----------+
```
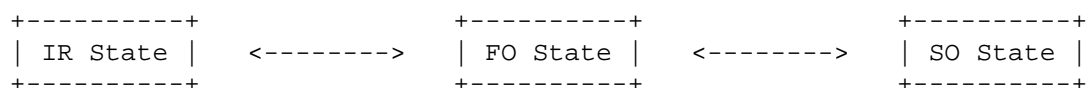
   Decisions about transitions between the various compression states
   are taken by the compressor on the basis of:

      - variations in packet headers
      - positive feedback from decompressor (Acknowledgments -- ACKs)
      - negative feedback from decompressor (Negative ACKs -- NACKs)
      - periodic timeouts (when operating in unidirectional mode, i.e.,
        over simplex channels or when feedback is not enabled)

How transitions are performed is explained in detail in chapter 5 for
each mode of operation.

## 4.3.1.1.  Initialization and Refresh (IR) State

The purpose of the IR state is to initialize the static parts of the
context at the decompressor or to recover after failure.  In this
state, the compressor sends complete header information.  This
includes all static and nonstatic fields in uncompressed form plus
some additional information.

The compressor stays in the IR state until it is fairly confident
that the decompressor has received the static information correctly.

## 4.3.1.2.  First Order (FO) State

The purpose of the FO state is to efficiently communicate
irregularities in the packet stream.  When operating in this state,
the compressor rarely sends information about all dynamic fields, and
the information sent is usually compressed at least partially.  Only
a few static fields can be updated.  The difference between IR and FO
should therefore be clear.

The compressor enters this state from the IR state, and from the SO
state whenever the headers of the packet stream do not conform to
their previous pattern.  It stays in the FO state until it is
confident that the decompressor has acquired all the parameters of
the new pattern.  Changes in fields that are always irregular are
communicated in all packets and are therefore part of what is a
uniform pattern.

Some or all packets sent in the FO state carry context updating
information.  It is very important to detect corruption of such
packets to avoid erroneous updates and context inconsistencies.

## 4.3.1.3.  Second Order (SO) State

This is the state where compression is optimal.  The compressor
enters the SO state when the header to be compressed is completely
predictable given the SN (RTP Sequence Number) and the compressor is
sufficiently confident that the decompressor has acquired all
parameters of the functions from SN to other fields.  Correct
decompression of packets sent in the SO state only hinges on correct
decompression of the SN.  However, successful decompression also
requires that the information sent in the preceding FO state packets
has been successfully received by the decompressor.

The compressor leaves this state and goes back to the FO state when
the header no longer conforms to the uniform pattern and cannot be
independently compressed on the basis of previous context
information.

## 4.3.2.  Decompressor states

The decompressor starts in its lowest compression state, "No Context"
and gradually transits to higher states.  The decompressor state
machine normally never leaves the "Full Context" state once it has
entered this state.

```
+--------------+         +----------------+         +--------------+
|  No Context  |  <--->  | Static Context |  <--->  | Full Context |
+--------------+         +----------------+         +--------------+
```
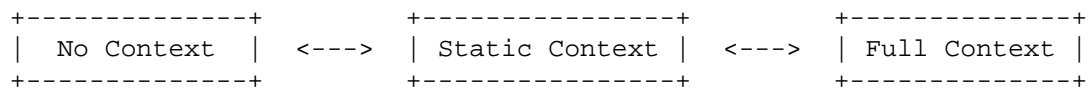
Initially, while working in the "No Context" state, the decompressor
has not yet successfully decompressed a packet.  Once a packet has
been decompressed correctly (for example, upon reception of an
initialization packet with static and dynamic information), the
decompressor can transit all the way to the "Full Context" state, and
only upon repeated failures will it transit back to lower states.
However, when that happens it first transits back to the "Static
Context" state.  There, reception of any packet sent in the FO state
is normally sufficient to enable transition to the "Full Context"
state again.  Only when decompression of several packets sent in the
FO state fails in the "Static Context" state will the decompressor go
all the way back to the "No Context" state.

When state transitions are performed is explained in detail in
chapter 5.

## 4.4.  Modes of operation

The ROHC scheme has three modes of operation, called Unidirectional,
Bidirectional Optimistic, and Bidirectional Reliable mode.

It is important to understand the difference between states, as
described in the previous chapter, and modes.  These abstractions are
orthogonal to each other.  The state abstraction is the same for all
modes of operation, while the mode controls the logic of state
transitions and what actions to perform in each state.

```
                    +---------------------+
                    |  Unidirectional Mode |
                    |   +--+  +--+  +--+   |
                    |   |IR|  |FO|  |SO|   |
                    |   +--+  +--+  +--+   |
                    +---------------------+
                       ^                 ^
                      /                   \
                     /                     \
                    v                       v
     +---------------------+     +---------------------+
     |   Optimistic Mode   |     |    Reliable Mode    |
     |   +--+  +--+  +--+   |     |   +--+  +--+  +--+   |
     |   |IR|  |FO|  |SO|   | <-------------> |   |IR|  |FO|  |SO|   |
     |   +--+  +--+  +--+   |     |   +--+  +--+  +--+   |
     +---------------------+     +---------------------+
```
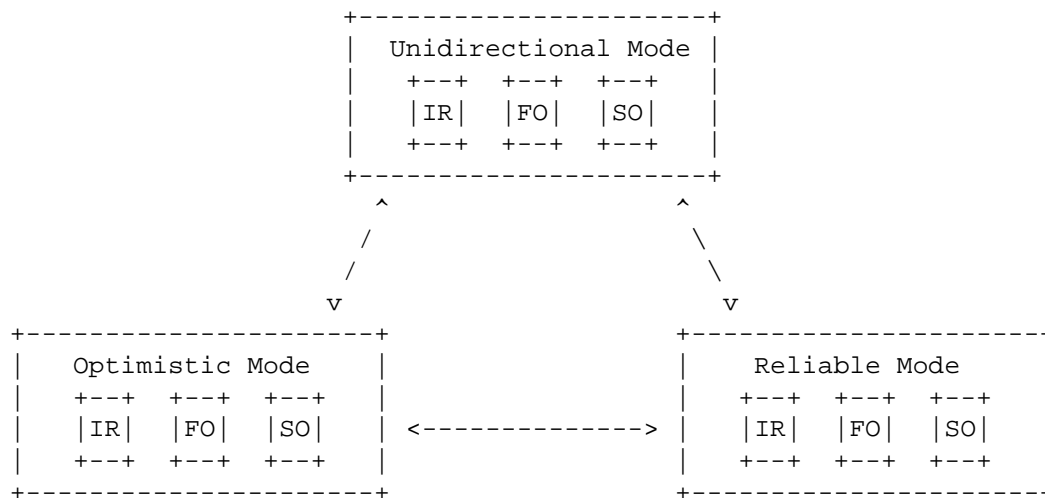
The optimal mode to operate in depends on the characteristics of the
environment of the compression protocol, such as feedback abilities,
error probabilities and distributions, effects of header size
variation, etc.  All ROHC implementations MUST implement and support
all three modes of operation.  The three modes are briefly described
in the following subsections.

Detailed descriptions of the three modes of operation regarding
compression and decompression logic are given in chapter 5.  The mode
transition mechanisms, too, are described in chapter 5.

4.4.1.  Unidirectional mode -- U-mode

When in the Unidirectional mode of operation, packets are sent in one
direction only: from compressor to decompressor.  This mode therefore
makes ROHC usable over links where a return path from decompressor to
compressor is unavailable or undesirable.

In U-mode, transitions between compressor states are performed only
on account of periodic timeouts and irregularities in the header
field change patterns in the compressed packet stream.  Due to the
periodic refreshes and the lack of feedback for initiation of error
recovery, compression in the Unidirectional mode will be less
efficient and have a slightly higher probability of loss propagation
compared to any of the Bidirectional modes.

Compression with ROHC MUST start in the Unidirectional mode.
Transition to any of the Bidirectional modes can be performed as soon
as a packet has reached the decompressor and it has replied with a
feedback packet indicating that a mode transition is desired (see
chapter 5).

4.4.2.  Bidirectional Optimistic mode -- O-mode

   The Bidirectional Optimistic mode is similar to the Unidirectional
   mode.  The difference is that a feedback channel is used to send
   error recovery requests and (optionally) acknowledgments of
   significant context updates from decompressor to compressor (not,
   however, for pure sequence number updates).  Periodic refreshes are
   not used in the Bidirectional Optimistic mode.

   O-mode aims to maximize compression efficiency and sparse usage of
   the feedback channel.  It reduces the number of damaged headers
   delivered to the upper layers due to residual errors or context
   invalidation.  The frequency of context invalidation may be higher
   than for R-mode, in particular when long loss/error bursts occur.
   Refer to section 4.7 for more details.

4.4.3.  Bidirectional Reliable mode -- R-mode

   The Bidirectional Reliable mode differs in many ways from the
   previous two.  The most important differences are a more intensive
   usage of the feedback channel and a stricter logic at both the
   compressor and the decompressor that prevents loss of context
   synchronization between compressor and decompressor except for very
   high residual bit error rates.  Feedback is sent to acknowledge all
   context updates, including updates of the sequence number field.
   However, not every packet updates the context in Reliable mode.

   R-mode aims to maximize robustness against loss propagation and
   damage propagation, i.e., minimize the probability of context
   invalidation, even under header loss/error burst conditions.  It may
   have a lower probability of context invalidation than O-mode, but a
   larger number of damaged headers may be delivered when the context
   actually is invalidated.  Refer to section 4.7 for more details.

4.5.  Encoding methods

   This chapter describes the encoding methods used for header fields.
   How the methods are applied to each field (e.g., values of associated
   parameters) is specified in section 5.7.

4.5.1. Least Significant Bits (LSB) encoding

   Least Significant Bits (LSB) encoding is used for header fields whose
   values are usually subject to small changes.  With LSB encoding, the
   k least significant bits of the field value are transmitted instead
   of the original field value, where k is a positive integer.  After
   receiving k bits, the decompressor derives the original value using a
   previously received value as reference (v_ref).

The scheme is guaranteed to be correct if the compressor and the
decompressor each use interpretation intervals

    1) in which the original value resides, and

    2) in which the original value is the only value that has the
       exact same k least significant bits as those transmitted.

The interpretation interval can be described as a function f(v_ref,
k).  Let

f(v_ref, k) = [v_ref - p, v_ref + (2^k - 1) - p]

where p is an integer.

```
    <------- interpretation interval (size is 2^k) ------->
    |------------+-------------------------------------|
  v_ref - p       v_ref                     v_ref + (2^k-1) - p
```
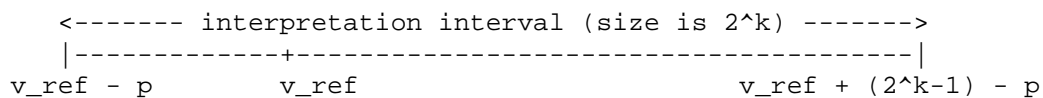
The function f has the following property: for any value k, the k
least significant bits will uniquely identify a value in f(v_ref, k).

The parameter p is introduced so that the interpretation interval can
be shifted with respect to v_ref.  Choosing a good value for p will
yield a more efficient encoding for fields with certain
characteristics.  Below are some examples:

a) For field values that are expected always to increase, p can be
   set to -1.  The interpretation interval becomes
   [v_ref + 1, v_ref + 2^k].

b) For field values that stay the same or increase, p can be set to
   0.  The interpretation interval becomes [v_ref, v_ref + 2^k - 1].

c) For field values that are expected to deviate only slightly from a
   constant value, p can be set to 2^(k-1) - 1.  The interpretation
   interval becomes [v_ref - 2^(k-1) + 1, v_ref + 2^(k-1)].

d) For field values that are expected to undergo small negative
   changes and larger positive changes, such as the RTP TS for video,
   or RTP SN when there is misordering, p can be set to 2^(k-2) - 1.
   The interval becomes [v_ref - 2^(k-2) + 1, v_ref + 3 * 2^(k-2)],
   i.e., 3/4 of the interval is used for positive changes.

The following is a simplified procedure for LSB compression and
decompression; it is modified for robustness and damage propagation
protection in the next subsection:

   1) The compressor (decompressor) always uses v_ref_c (v_ref_d), the
      last value that has been compressed (decompressed), as v_ref;

   2) When compressing a value v, the compressor finds the minimum value
      of k such that v falls into the interval f(v_ref_c, k).  Call this
      function k = g(v_ref_c, v). When only a few distinct values of k
      are possible, for example due to limitations imposed by packet
      formats (see section 5.7), the compressor will instead pick the
      smallest k that puts v in the interval f(v_ref_c, k).

   3) When receiving m LSBs, the decompressor uses the interpretation
      interval f(v_ref_d, m), called interval_d.  It picks as the
      decompressed value the one in interval_d whose LSBs match the
      received m bits.

   Note that the values to be encoded have a finite range; for example,
   the RTP SN ranges from 0 to 0xFFFF.  When the SN value is close to 0
   or 0xFFFF, the interpretation interval can straddle the wraparound
   boundary between 0 and 0xFFFF.

   The scheme is complicated by two factors: packet loss between the
   compressor and decompressor, and transmission errors undetected by
   the lower layer.  In the former case, the compressor and decompressor
   will lose the synchronization of v_ref, and thus also of the
   interpretation interval.  If v is still covered by the
   intersection(interval_c, interval_d), the decompression will be
   correct.  Otherwise, incorrect decompression will result.  The next
   section will address this issue further.

   In the case of undetected transmission errors, the corrupted LSBs
   will give an incorrectly decompressed value that will later be used
   as v_ref_d, which in turn is likely to lead to damage propagation.
   This problem is addressed by using a secure reference, i.e., a
   reference value whose correctness is verified by a protecting CRC.
   Consequently, the procedure 1) above is modified as follows:

   1) a) the compressor always uses as v_ref_c the last value that has
         been compressed and sent with a protecting CRC.
      b) the decompressor always uses as v_ref_d the last correct
         value, as verified by a successful CRC.

   Note that in U/O-mode, 1) b) is modified so that if decompression of
   the SN fails using the last verified SN reference, another
   decompression attempt is made using the last but one verified SN
   reference.  This procedure mitigates damage propagation when a small
   CRC fails to detect a damaged value.  See section 5.3.2.2.3 for
   further details.

4.5.2.  Window-based LSB encoding (W-LSB encoding)

   This section describes how to modify the simplified algorithm in
   4.5.1 to achieve robustness.

   The compressor may not be able to determine the exact value of
   v_ref_d that will be used by the decompressor for a particular value
   v, since some candidates for v_ref_d may have been lost or damaged.
   However, by using feedback or by making reasonable assumptions, the
   compressor can limit the candidate set.  The compressor then
   calculates k such that no matter which v_ref_d in the candidate set
   the decompressor uses, v is covered by the resulting interval_d.

   Since the decompressor always uses as the reference the last received
   value where the CRC succeeded, the compressor maintains a sliding
   window containing the candidates for v_ref_d.  The sliding window is
   initially empty.  The following operations are performed on the
   sliding window by the compressor:

   1) After sending a value v (compressed or uncompressed) protected by
      a CRC, the compressor adds v to the sliding window.

   2) For each value v being compressed, the compressor chooses k =
      max(g(v_min, v), g(v_max, v)), where v_min and v_max are the
      minimum and maximum values in the sliding window, and g is the
      function defined in the previous section.

   3) When the compressor is sufficiently confident that a certain value
      v and all values older than v will not be used as reference by the
      decompressor, the window is advanced by removing those values
      (including v).  The confidence may be obtained by various means.
      In R-mode, an ACK from the decompressor implies that values older
      than the ACKed one can be removed from the sliding window.  In
      U/O-mode there is always a CRC to verify correct decompression,
      and a sliding window with a limited maximum width is used.  The
      window width is an implementation dependent optimization
      parameter.

   Note that the decompressor follows the procedure described in the
   previous section, except that in R-mode it MUST ACK each header
   received with a succeeding CRC (see also section 5.5).

4.5.3. Scaled RTP Timestamp encoding

   The RTP Timestamp (TS) will usually not increase by an arbitrary
   number from packet to packet.  Instead, the increase is normally an
   integral multiple of some unit (TS_STRIDE).  For example, in the case
   of audio, the sample rate is normally 8 kHz and one voice frame may

cover 20 ms.  Furthermore, each voice frame is often carried in one
RTP packet.  In this case, the RTP increment is always n * 160 (=
8000 * 0.02), for some integer n.  Note that silence periods have no
impact on this, as the sample clock at the source normally keeps
running without changing either frame rate or frame boundaries.

In the case of video, there is usually a TS_STRIDE as well when the
video frame level is considered.  The sample rate for most video
codecs is 90 kHz.  If the video frame rate is fixed, say, to 30
frames/second, the TS will increase by n * 3000 (= n * 90000 / 30)
between video frames.  Note that a video frame is often divided into
several RTP packets to increase robustness against packet loss.  In
this case several RTP packets will carry the same TS.

When using scaled RTP Timestamp encoding, the TS is downscaled by a
factor of TS_STRIDE before compression.  This saves

    floor(log2(TS_STRIDE))

bits for each compressed TS.  TS and TS_SCALED satisfy the following
equality:

    TS = TS_SCALED * TS_STRIDE + TS_OFFSET

TS_STRIDE is explicitly, and TS_OFFSET implicitly, communicated to
the decompressor.  The following algorithm is used:

1. Initialization: The compressor sends to the decompressor the value
   of TS_STRIDE and the absolute value of one or several TS fields.
   The latter are used by the decompressor to initialize TS_OFFSET to
   (absolute value) modulo TS_STRIDE.  Note that TS_OFFSET is the
   same regardless of which absolute value is used, as long as the
   unscaled TS value does not wrap around; see 4) below.

2. Compression: After initialization, the compressor no longer
   compresses the original TS values.  Instead, it compresses the
   downscaled values: TS_SCALED = TS / TS_STRIDE.  The compression
   method could be either W-LSB encoding or the timer-based encoding
   described in the next section.

3. Decompression: When receiving the compressed value of TS_SCALED,
   the decompressor first derives the value of the original
   TS_SCALED.  The original RTP TS is then calculated as TS =
   TS_SCALED * TS_STRIDE + TS_OFFSET.

4. Offset at wraparound: Wraparound of the unscaled 32-bit TS will
   invalidate the current value of TS_OFFSET used in the equation
   above.  For example, let us assume TS_STRIDE = 160 = 0xA0 and the

current TS = 0xFFFFFFF0.  TS_OFFSET is then 0x50 = 80.  Then if
the next RTP TS = 0x00000130 (i.e., the increment is 160 * 2 =
320), the new TS_OFFSET should be 0x00000130 modulo 0xA0 = 0x90 =
144.  The compressor is not required to re-initialize TS_OFFSET at
wraparound.  Instead, the decompressor MUST detect wraparound of
the unscaled TS (which is trivial) and update TS_OFFSET to

   TS_OFFSET = (Wrapped around unscaled TS) modulo TS_STRIDE

5. Interpretation interval at wraparound: Special rules are needed
   for the interpretation interval of the scaled TS at wraparound,
   since the maximum scaled TS, TSS_MAX, (0xFFFFFFFF / TS_STRIDE) may
   not have the form $2^m - 1$.  For example, when TS_STRIDE is 160,
   the scaled TS is at most 26843545 which has LSBs 10011001.  The
   wraparound boundary between the TSS_MAX may thus not correspond to
   a natural boundary between LSBs.

```
            interpretation interval
     |<----------------------------->|

                  unused                        scaled TS
------------|--------------|---------------------->
               TSS_MAX          zero
```
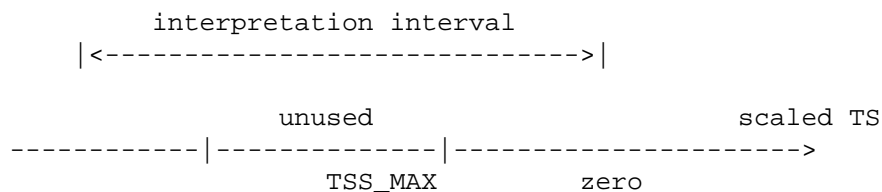
When TSS_MAX is part of the interpretation interval, a number of
unused values are inserted into it after TSS_MAX such that their
LSBs follow naturally upon each other.  For example, for TS_STRIDE
= 160 and k = 4, values corresponding to the LSBs 1010 through
1111 are inserted.  The number of inserted values depends on k and
the LSBs of the maximum scaled TS.  The number of valid values in
the interpretation interval should be high enough to maintain
robustness.  This can be ensured by the following rule:

     Let a be the number of LSBs needed if there was no
     wraparound, and let b be the number of LSBs needed to
     disambiguate between TSS_MAX and zero where the a LSBs of
     TSS_MAX are set to zero.  The number of LSB bits to send
     while TSS_MAX or zero is part of the interpretation interval
     is b.

This scaling method can be applied to many frame-based codecs.
However, the value of TS_STRIDE might change during a session, for
example as a result of adaptation strategies.  If that happens, the
unscaled TS is compressed until re-initialization of the new
TS_STRIDE and TS_OFFSET is completed.

4.5.4.  Timer-based compression of RTP Timestamp

   The RTP Timestamp [RFC 1889] is defined to identify the number of the
   first sample used to generate the payload.  When 1) RTP packets carry
   payloads corresponding to a fixed sampling interval, 2) the sampling
   is done at a constant rate, and 3) packets are generated in lock-step
   with sampling, then the timestamp value will closely approximate a
   linear function of the time of day.  This is the case for
   conversational media, such as interactive speech.  The linear ratio
   is determined by the source sample rate.  The linear pattern can be
   complicated by packetization (e.g., in the case of video where a
   video frame usually corresponds to several RTP packets) or frame
   rearrangement (e.g., B-frames are sent out-of-order by some video
   codecs).

   With a fixed sample rate of 8 kHz, 20 ms in the time domain is
   equivalent to an increment of 160 in the unscaled TS domain, and to
   an increment of 1 in the scaled TS domain with TS_STRIDE = 160.

   As a consequence, the (scaled) TS of headers arriving at the
   decompressor will be a linear function of time of day, with some
   deviation due to the delay jitter (and the clock inaccuracies)
   between the source and the decompressor.  In normal operation, i.e.,
   no crashes or failures, the delay jitter will be bounded to meet the
   requirements of conversational real-time traffic.  Hence, by using a
   local clock the decompressor can obtain an approximation of the
   (scaled) TS in the header to be decompressed by considering its
   arrival time.  The approximation can then be refined with the k LSBs
   of the (scaled) TS carried in the header.  The value of k required to
   ensure correct decompression is a function of the jitter between the
   source and the decompressor.

   If the compressor knows the potential jitter introduced between
   compressor and decompressor, it can determine k by using a local
   clock to estimate jitter in packet arrival times, or alternatively it
   can use a fixed k and discard packets arriving too much out of time.

   The advantages of this scheme include:

   a) The size of the compressed TS is constant and small.  In
      particular, it does NOT depend on the length of silence intervals.
      This is in contrast to other TS compression techniques, which at
      the beginning of a talkspurt require sending a number of bits
      dependent on the duration of the preceding silence interval.

   b) No synchronization is required between the clock local to the
      compressor and the clock local to the decompressor.

Note that although this scheme can be made to work using both scaled
and unscaled TS, in practice it is always combined with scaled TS
encoding because of the less demanding requirement on the clock
resolution, e.g., 20 ms instead of 1/8 ms.  Therefore, the algorithm
described below assumes that the clock-based encoding scheme operates
on the scaled TS.  The case of unscaled TS would be similar, with
changes to scale factors.

The major task of the compressor is to determine the value of k.  Its
sliding window now contains not only potential reference values for
the TS but also their times of arrival at the compressor.

1) The compressor maintains a sliding window

   {(T_j, a_j), for each header j that can be used as a reference},

   where T_j is the scaled TS for header j, and a_j is the arrival
   time of header j.  The sliding window serves the same purpose as
   the W-LSB sliding window of section 4.5.2.

2) When a new header n arrives with T_n as the scaled TS, the
   compressor notes the arrival time a_n.  It then calculates

      Max_Jitter_BC =

         max {|(T_n - T_j) - ((a_n - a_j) / TIME_STRIDE)|,
             for all headers j in the sliding window},

   where TIME_STRIDE is the time interval equivalent to one
   TS_STRIDE, e.g., 20 ms.  Max_Jitter_BC is the maximum observed
   jitter before the compressor, in units of TS_STRIDE, for the
   headers in the sliding window.

3) k is calculated as

         k = ceiling(log2(2 * J + 1),

      where J = Max_Jitter_BC + Max_Jitter_CD + 2.

   Max_Jitter_CD is the upper bound of jitter expected on the
   communication channel between compressor and decompressor (CD-CC).
   It depends only on the characteristics of CD-CC.

The constant 2 accounts for the quantization error introduced by
the clocks at the compressor and decompressor, which can be +/-1.

Note that the calculation of k follows the compression algorithm
described in section 4.5.1, with p = 2^(k-1) - 1.

4) The sliding window is subject to the same window operations as in
   section 4.5.2, 1) and 3), except that the values added and removed
   are paired with their arrival times.

Decompressor:

1) The decompressor uses as its reference header the last correctly
   (as verified by CRC) decompressed header.  It maintains the pair
   (T_ref, a_ref), where T_ref is the scaled TS of the reference
   header, and a_ref is the arrival time of the reference header.

2) When receiving a compressed header n at time a_n, the
   approximation of the original scaled TS is calculated as:

       T_approx = T_ref + (a_n - a_ref) / TIME_STRIDE.

3) The approximation is then refined by the k least significant bits
   carried in header n, following the decompression algorithm of
   section 4.5.1, with p = 2^(k-1) - 1.

   Note: The algorithm does not assume any particular pattern in the
   packets arriving at the compressor, i.e., it tolerates reordering
   before the compressor and nonincreasing RTP Timestamp behavior.

   Note: Integer arithmetic is used in all equations above.  If
   TIME_STRIDE is not equal to an integral number of clock ticks,
   time must be normalized such that TIME_STRIDE is an integral
   number of clock ticks.  For example, if a clock tick is 20 ms and
   TIME_STRIDE is 30 ms, (a_n - a_ref) in 2) can be multiplied by 3
   and TIME_STRIDE can have the value 2.

   Note: The clock resolution of the compressor or decompressor can
   be worse than TIME_STRIDE, in which case the difference, i.e.,
   actual resolution - TIME_STRIDE, is treated as additional jitter
   in the calculation of k.

   Note: The clock resolution of the decompressor may be communicated
   to the compressor using the CLOCK feedback option.

   Note: The decompressor may observe the jitter and report this to
   the compressor using the JITTER feedback option.  The compressor
   may use this information to refine its estimate of Max_Jitter_CD.

4.5.5.  Offset IP-ID encoding

   As all IPv4 packets have an IP Identifier to allow for fragmentation,
   ROHC provides for transparent compression of this ID.  There is no
   explicit support in ROHC for the IPv6 fragmentation header, so there
   is never a need to discuss IP IDs outside the context of IPv4.

   This section assumes (initially) that the IPv4 stack at the source
   host assigns IP-ID according to the value of a 2-byte counter which
   is increased by one after each assignment to an outgoing packet.
   Therefore, the IP-ID field of a particular IPv4 packet flow will
   increment by 1 from packet to packet except when the source has
   emitted intermediate packets not belonging to that flow.

   For such IPv4 stacks, the RTP SN will increase by 1 for each packet
   emitted and the IP-ID will increase by at least the same amount.
   Thus, it is more efficient to compress the offset, i.e., (IP-ID - RTP
   SN), instead of IP-ID itself.

   The remainder of section 4.5.5 describes how to compress/decompress
   the sequence of offsets using W-LSB encoding/decoding, with p = 0
   (see section 4.5.1).  All IP-ID arithmetic is done using unsigned
   16-bit quantities, i.e., modulo $2^{16}$.

   Compressor:

      The compressor uses W-LSB encoding (section 4.5.2) to compress a
      sequence of offsets

         $Offset_i = ID_i - SN_i$,

      where $ID_i$ and $SN_i$ are the values of the IP-ID and RTP SN of
      header i.  The sliding window contains such offsets and not the
      values of header fields, but the rules for adding and deleting
      offsets from the window otherwise follow section 4.5.2.

   Decompressor:

      The reference header is the last correctly (as verified by CRC)
      decompressed header.

      When receiving a compressed packet m, the decompressor calculates
      $Offset_{ref} = ID_{ref} - SN_{ref}$, where $ID_{ref}$ and $SN_{ref}$ are the
      values of IP-ID and RTP SN in the reference header, respectively.

Then W-LSB decoding is used to decompress Offset_m, using the
received LSBs in packet m and Offset_ref.  Note that m may contain
zero LSBs for Offset_m, in which case Offset_m = Offset_ref.

Finally, the IP-ID for packet m is regenerated as

IP-ID for m = decompressed SN of packet m + Offset_m

Network byte order:

Some IPv4 stacks do use a counter to generate IP ID values as
described, but do not transmit the contents of this counter in
network byte order, but instead send the two octets reversed.  In
this case, the compressor can compress the IP-ID field after
swapping the bytes.  Consequently, the decompressor also swaps the
bytes of the IP-ID after decompression to regenerate the original
IP-ID.  This requires that the compressor and the decompressor
synchronize on the byte order of the IP-ID field using the NBO or
NBO2 flag (see section 5.7).

Random IP Identifier:

Some IPv4 stacks generate the IP Identifier values using a
pseudo-random number generator.  While this may provide some
security benefits, it makes it pointless to attempt compressing
the field.  Therefore, the compressor should detect such random
behavior of the field.  After detection and synchronization with
the decompressor using the RND or RND2 flag, the field is sent
as-is in its entirety as additional octets after the compressed
header.

4.5.6.  Self-describing variable-length values

The values of TS_STRIDE and a few other compression parameters can
vary widely.  TS_STRIDE can be 160 for voice and 90 000 for 1 f/s
video.  To optimize the transfer of such values, a variable number of
octets is used to encode them.  The number of octets used is
determined by the first few bits of the first octet:

First bit is 0: 1 octet.
        7 bits transferred.
        Up to 127 decimal.
        Encoded octets in hexadecimal: 00 to 7F

First bits are 10: 2 octets.
        14 bits transferred.
        Up to 16 383 decimal.
        Encoded octets in hexadecimal: 80 00 to BF FF

   First bits are 110: 3 octets.
            21 bits transferred.
            Up to 2 097 151 decimal.
            Encoded octets in hexadecimal: C0 00 00 to DF FF FF

   First bits are 111: 4 octets.
            29 bits transferred.
            Up to 536 870 911 decimal.
            Encoded octets in hexadecimal: E0 00 00 00 to FF FF FF FF

4.5.7.  Encoded values across several fields in compressed headers

   When a compressed header has an extension, pieces of an encoded value
   can be present in more than one field.  When an encoded value is
   split over several fields in this manner, the more significant bits
   of the value are closer to the beginning of the header.  If the
   number of bits available in compressed header fields exceeds the
   number of bits in the value, the most significant field is padded
   with zeroes in its most significant bits.

   For example, an unscaled TS value can be transferred using an UOR-2
   header (see section 5.7) with an extension of type 3.  The Tsc bit of
   the extension is then unset (zero) and the variable length TS field
   of the extension is 4 octets, with 29 bits available for the TS (see
   section 4.5.6).  The UOR-2 TS field will contain the three most
   significant bits of the unscaled TS, and the 4-octet TS field in the
   extension will contain the remaining 29 bits.

4.6.  Errors caused by residual errors

   ROHC is designed under the assumption that packets can be damaged
   between the compressor and decompressor, and that such damaged
   packets can be delivered to the decompressor ("residual errors").

   Residual errors may damage the SN in compressed headers.  Such damage
   will cause generation of a header which upper layers may not be able
   to distinguish from a correct header.  When the compressed header
   contains a CRC, the CRC will catch the bad header with a probability
   dependent on the size of the CRC.  When ROHC does not detect the bad
   header, it will be delivered to upper layers.

   Damage is not confined to the SN:

   a) Damage to packet type indication bits can cause a header to be
      interpreted as having a different packet type.

   b) Damage to CID information may cause a packet to be interpreted
      according to another context and possibly also according to
      another profile.  Damage to CIDs will be more harmful when a large
      part of the CID space is being used, so that it is likely that the
      damaged CID corresponds to an active context.

   c) Feedback information can also be subject to residual errors, both
      when feedback is piggybacked and when it is sent in separate ROHC
      packets.  ROHC uses sanity checks and adds CRCs to vital feedback
      information to allow detection of some damaged feedback.

      Note that context damage can also result in generation of
      incorrect headers; section 4.7 elaborates further on this.

4.7.  Impairment considerations

   Impairments to headers can be classified into the following types:

      (1) the lower layer was not able to decode the packet and did not
          deliver it to ROHC,

      (2) the lower layer was able to decode the packet, but discarded
          it because of a detected error,

      (3) ROHC detected an error in the generated header and discarded
          the packet, or

      (4) ROHC did not detect that the regenerated header was damaged
          and delivered it to upper layers.

   Impairments cause loss or damage of individual headers.  Some
   impairment scenarios also cause context invalidation, which in turn
   results in loss propagation and damage propagation.  Damage
   propagation and undetected residual errors both contribute to the
   number of damaged headers delivered to upper layers.  Loss
   propagation and impairments resulting in loss or discarding of single
   packets both contribute to the packet loss seen by upper layers.

   Examples of context invalidating scenarios are:

      (a) Impairment of type (4) on the forward channel, causing the
          decompressor to update its context with incorrect information;

(b) Loss/error burst of pattern update headers: Impairments of
    types (1),(2) and (3) on consecutive pattern update headers; a
    pattern update header is a header carrying a new pattern
    information, e.g., at the beginning of a new talk spurt; this
    causes the decompressor to lose the pattern update
    information;

(c) Loss/error burst of headers: Impairments of types (1),(2) and
    (3) on a number of consecutive headers that is large enough to
    cause the decompressor to lose the SN synchronization;

(d) Impairment of type (4) on the feedback channel which mimics a
    valid ACK and makes the compressor update its context;

(e) a burst of damaged headers (3) erroneously triggers the "k-
    out-of-n" rule for detecting context invalidation, which
    results in a NACK/update sequence during which headers are
    discarded.

Scenario (a) is mitigated by the CRC carried in all context updating
headers.  The larger the CRC, the lower the chance of context
invalidation caused by (a).  In R-mode, the CRC of context updating
headers is always 7 bits or more.  In U/O-mode, it is usually 3 bits
and sometimes 7 or 8 bits.

Scenario (b) is almost completely eliminated when the compressor
ensures through ACKs that no context updating headers are lost, as in
R-mode.

Scenario (c) is almost completely eliminated when the compressor
ensures through ACKs that the decompressor will always detect the SN
wraparound, as in R-mode.  It is also mitigated by the SN repair
mechanisms in U/O-mode.

Scenario (d) happens only when the compressor receives a damaged
header that mimics an ACK of some header present in the W-LSB window,
say ACK of header 2, while in reality header 2 was never received or
accepted by the decompressor, i.e., header 2 was subject to
impairment (1), (2) or (3).  The damaged header must mimic the
feedback packet type, the ACK feedback type, and the SN LSBs of some
header in the W-LSB window.

Scenario (e) happens when a burst of residual errors causes the CRC
check to fail in k out of the last n headers carrying CRCs.  Large k
and n reduces the probability of scenario (e), but also increases the
number of headers lost or damaged as a consequence of any context
invalidation.

ROHC detects damaged headers using CRCs over the original headers.
The smallest headers in this document either include a 3-bit CRC
(U/O-mode) or do not include a CRC (R-mode).  For the smallest
headers, damage is thus detected with a probability of roughly 7/8
for U/O-mode.  For R-mode, damage to the smallest headers is not
detected.

All other things (coding scheme at lower layers, etc.) being equal,
the rate of headers damaged by residual errors will be lower when
headers are compressed compared when they are not, since fewer bits
are transmitted.  Consequently, for a given ROHC CRC setup the rate
of incorrect headers delivered to applications will also be reduced.

The above analysis suggests that U/O-mode may be more prone than R-
mode to context invalidation.  On the other hand, the CRC present in
all U/O-mode headers continuously screens out residual errors coming
from lower layers, reduces the number of damaged headers delivered to
upper layers when context is invalidated, and permits quick detection
of context invalidation.

R-mode always uses a stronger CRC on context updating headers, but no
CRC in other headers.  A residual error on a header which carries no
CRC will result in a damaged header being delivered to upper layers
(4).  The number of damaged headers delivered to the upper layers
depends on the ratio of headers with CRC vs. headers without CRC,
which is a compressor parameter.

5.  The protocol

5.1.  Data structures

The ROHC protocol is based on a number of parameters that form part
of the negotiated channel state and the per-context state.  This
section describes some of this state information in an abstract way.
Implementations can use a different structure for and representation
of this state.  In particular, negotiation protocols that set up the
per-channel state need to establish the information that constitutes
the negotiated channel state, but it is not necessary to exchange it
in the form described here.

5.1.1.  Per-channel parameters

MAX_CID: Nonnegative integer; highest context ID number to be used by
the compressor (note that this parameter is not coupled to, but in
effect further constrained by, LARGE_CIDS).

LARGE_CIDS: Boolean; if false, the short CID representation (0 bytes
or 1 prefix byte, covering CID 0 to 15) is used; if true, the
embedded CID representation (1 or 2 embedded CID bytes covering CID 0
to 16383) is used.

PROFILES: Set of nonnegative integers, each integer indicating a
profile supported by the decompressor.  The compressor MUST NOT
compress using a profile not in PROFILES.

FEEDBACK_FOR: Optional reference to a channel in the reverse
direction.  If provided, this parameter indicates which channel any
feedback sent on this channel refers to (see 5.7.6.1).

MRRU: Maximum reconstructed reception unit.  This is the size of the
largest reconstructed unit in octets that the decompressor is
expected to reassemble from segments (see 5.2.5).  Note that this
size includes the CRC.  If MRRU is negotiated to be 0, no segment
headers are allowed on the channel.

5.1.2.  Per-context parameters, profiles

Per-context parameters are established with IR headers (see section
5.2.3).  An IR header contains a profile identifier, which determines
how the rest of the header is to be interpreted.  Note that the
profile parameter determines the syntax and semantics of the packet
type identifiers and packet types used in conjunction with a specific
context.  This document describes profiles 0x0000, 0x0001, 0x0002,
and 0x0003; further profiles may be defined when ROHC is extended in
the future.

Profile 0x0000 is for sending uncompressed IP packets.  See section
   5.10.

Profile 0x0001 is for RTP/UDP/IP compression, see sections 5.3
   through 5.9.

Profile 0x0002 is for UDP/IP compression, i.e., compression of the
   first 12 octets of the UDP payload is not attempted.  See section
   5.11.

Profile 0x0003 is for ESP/IP compression, i.e., compression of the
   header chain up to and including the first ESP header, but not
   subsequent subheaders.  See section 5.12.

Initially, all contexts are in no context state, i.e., all packets
referencing this context except IR packets are discarded.  If defined
by a "ROHC over X" document, per-channel negotiation can be used to
pre-establish state information for a context (e.g., negotiating

profile 0x0000 for CID 15).  Such state information can also be
marked read-only in the negotiation, which would cause the
decompressor to discard any IR packet attempting to modify it.

## 5.1.3.  Contexts and context identifiers

Associated with each compressed flow is a context, which is the state
compressor and decompressor maintain in order to correctly compress
or decompress the headers of the packet stream.  Contexts are
identified by a context identifier, CID, which is sent along with
compressed headers and feedback information.

The CID space is distinct for each channel, i.e., CID 3 over channel
A and CID 3 over channel B do not refer to the same context, even if
the endpoints of A and B are the same nodes.  In particular, CIDs for
any pairs of forward and reverse channels are not related (forward
and reverse channels need not even have CID spaces of the same size).

Context information is conceptually kept in a table.  The context
table is indexed using the CID which is sent along with compressed
headers and feedback information.  The CID space can be negotiated to
be either small, which means that CIDs can take the values 0 through
15, or large, which means that CIDs take values between 0 and $2^{14} -
1 = 16383$.  Whether the CID space is large or small is negotiated no
later than when a channel is established.

A small CID with the value 0 is represented using zero bits.  A small
CID with a value from 1 to 15 is represented by a four-bit field in
place of a packet type field (Add-CID) plus four more bits.  A large
CID is represented using the encoding scheme of section 4.5.6,
limited to two octets.

## 5.2.  ROHC packets and packet types

The packet type indication scheme for ROHC has been designed under
the following constraints:

a) it must be possible to use only a limited number of packet sizes;
b) it must be possible to send feedback information in separate ROHC
   packets as well as piggybacked on forward packets;
c) it is desirable to allow elimination of the CID for one packet
   stream when few packet streams share a channel;
d) it is anticipated that some packets with large headers may be
   larger than the MTU of very constrained lower layers.

These constraints have led to a design which includes

- optional padding,
- a feedback packet type,
- an optional Add-CID octet which provides 4 bits of CID, and
- a simple segmentation and reassembly mechanism.

A ROHC packet has the following general format (in the diagram,
colons ":" indicate that the part is optional):

```
 --- --- --- --- --- --- --- ---
:           Padding              :  variable length
 --- --- --- --- --- --- --- ---
:           Feedback             :  0 or more feedback elements
 --- --- --- --- --- --- --- ---
:            Header              :  variable, with CID information
 --- --- --- --- --- --- --- ---
:            Payload             :
 --- --- --- --- --- --- --- ---
```

Padding is any number (zero or more) of padding octets.  Either of
Feedback or Header must be present.

Feedback elements always start with a packet type indication.
Feedback elements carry internal CID information.  Feedback is
described in section 5.2.2.

Header is either a profile-specific header or an IR or IR-DYN header
(see sections 5.2.3 and 5.2.4).  Header either

1) does not carry any CID information (indicating CID zero), or
2) includes one Add-CID Octet (see below), or
3) contains embedded CID information of length one or two octets.

Alternatives 1) and 2) apply only to compressed headers in channels
where the CID space is small.  Alternative 3) applies only to
compressed headers in channels where the CID space is large.

Padding Octet

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   1   1   0   0   0   0   0 |
+---+---+---+---+---+---+---+---+
```

Add-CID Octet

```
    0   1   2   3   4   5   6   7
  +---+---+---+---+---+---+---+---+
  | 1   1   1   0 |      CID      |
  +---+---+---+---+---+---+---+---+
```

CID:    0x1 through 0xF indicates CIDs 1 through 15.

Note: The Padding Octet looks like an Add-CID octet for CID 0.

Header either starts with a packet type indication or has a packet
type indication immediately following an Add-CID Octet.  All Header
packet types have the following general format (in the diagram,
slashes "/" indicate variable length):

```
    0                 x-1  x         7
   --- --- --- --- --- --- --- ---
  :          Add-CID octet        :  if (CID 1-15) and (small CIDs)
  +---+--- --- --- ---+--- --- ---+
  | type indication   |  body     |  1 octet (8-x bits of body)
  +---+--- ---+---+---+--- --- ---+
  :                               :
  /    0, 1, or 2 octets of CID   /  1 or 2 octets if (large CIDs)
  :                               :
  +---+---+---+---+---+---+---+---+
  /             body              /  variable length
  +---+---+---+---+---+---+---+---+
```

The large CID, if present, is encoded according to section 4.5.6.

5.2.1.  ROHC feedback

Feedback carries information from decompressor to compressor.  The
following principal kinds of feedback are supported.  In addition to
the kind of feedback, other information may be included in profile-
specific feedback information.

ACK         : Acknowledges successful decompression of a packet,
              which means that the context is up-to-date with a high
              probability.

NACK        : Indicates that the dynamic context of the
              decompressor is out of sync.  Generated when several
              successive packets have failed to be decompressed
              correctly.

     STATIC-NACK : Indicates that the static context of the decompressor
                   is not valid or has not been established.

   It is anticipated that feedback to the compressor can be realized in
   many ways, depending on the properties of the particular lower layer.
   The exact details of how feedback is realized is to be specified in a
   "ROHC over X" document, for each lower layer X in question.  For
   example, feedback might be realized using

   1) lower-layer specific mechanisms

   2) a dedicated feedback-only channel, realized for example by the
      lower layer providing a way to indicate that a packet is a
      feedback packet

   3) a dedicated feedback-only channel, where the timing of the
      feedback provides information about which compressed packet caused
      the feedback

   4) interspersing of feedback packets among normal compressed packets
      going in the same direction as the feedback (lower layers do not
      indicate feedback)

   5) piggybacking of feedback information in compressed packets going
      in the same direction as the feedback (this technique may reduce
      the per-feedback overhead)

   6) interspersing and piggybacking on the same channel, i.e., both 4)
      and 5).

   Alternatives 1-3 do not place any particular requirements on the ROHC
   packet type scheme.  Alternatives 4-6 do, however.  The ROHC packet
   type scheme has been designed to allow alternatives 4-6 (these may be
   used for example over PPP):

   a) The ROHC scheme provides a feedback packet type.  The packet type
      is able to carry variable-length feedback information.

   b) The feedback information sent on a particular channel is passed
      to, and interpreted by, the compressor associated with feedback on
      that channel.  Thus, the feedback information must contain CID
      information if the associated compressor can use more than one
      context.  The ROHC feedback scheme requires that a channel carries
      feedback to at most one compressor.  How a compressor is
      associated with feedback on a particular channel needs to be
      defined in a "ROHC over X" document.

   c) The ROHC feedback information format is octet-aligned, i.e.,
      starts at an octet boundary, to allow using the format over a
      dedicated feedback channel, 2).

   d) To allow piggybacking, 5), it is possible to deduce the length of
      feedback information by examining the first few octets of the
      feedback.  This allows the decompressor to pass piggybacked
      feedback information to the associated same-side compressor
      without understanding its format.  The length information
      decouples the decompressor from the compressor in the sense that
      the decompressor can process the compressed header immediately
      without waiting for the compressor to hand it back after parsing
      the feedback information.

5.2.2.  ROHC feedback format

   Feedback sent on a ROHC channel consists of one or more concatenated
   feedback elements, where each feedback element has the following
   format:

```
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   1   1   1   0 |   Code    |  feedback type octet
   +---+---+---+---+---+---+---+---+
   :             Size              :  if Code = 0
   +---+---+---+---+---+---+---+---+
   /          feedback data        /  variable length
   +---+---+---+---+---+---+---+---+
```

   Code: 0 indicates that a Size octet is present.
         1-7 indicates the size of the feedback data field in
         octets.

   Size: Optional octet indicating the size of the feedback data
         field in octets.

   feedback data: Profile-specific feedback information.  Includes
         CID information.

   The total size of the feedback data field is determinable upon
   reception by the decompressor, by inspection of the Code field and
   possibly the Size field.  This explicit length information allows
   piggybacking and also sending more than one feedback element in a
   packet.

   When the decompressor has determined the size of the feedback data
   field, it removes the feedback type octet and the Size field (if
   present) and hands the rest to the same-side associated compressor

together with an indication of the size.  The feedback data received
by the compressor has the following structure (feedback sent on a
dedicated feedback channel MAY also use this format):

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
:          Add-CID octet        :  if for small CIDs and (CID != 0)
+---+---+---+---+---+---+---+---+
:                              :
/  large CID (4.5.6 encoding)  /  1-2 octets if for large CIDs
:                              :
+---+---+---+---+---+---+---+---+
/           feedback           /
+---+---+---+---+---+---+---+---+
```

The large CID, if present, is encoded according to section 4.5.6.
CID information in feedback data indicates the CID of the packet
stream for which feedback is sent.  Note that the LARGE_CIDS
parameter that controls whether a large CID is present is taken from
the channel state of the receiving compressor's channel, NOT from
that of the channel carrying the feedback.

It is REQUIRED that the feedback field have either of the following
two formats:

FEEDBACK-1

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| profile specific information  |  1 octet
+---+---+---+---+---+---+---+---+
```

FEEDBACK-2

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|Acktype|                       |
+---+---+   profile specific    /  at least 2 octets
/            information        |
+---+---+---+---+---+---+---+---+
```

Acktype:  0 = ACK
          1 = NACK
          2 = STATIC-NACK
          3 is reserved (MUST NOT be used.  Otherwise unparseable.)

The compressor can use the following logic to parse the feedback
field.

   1) If for large CIDs, the feedback will always start with a CID
      encoded according to section 4.5.6.  If the first bit is 0, the
      CID uses one octet.  If the first bit is 1, the CID uses two
      octets.

   2) If for small CIDs, and the size is one octet, the feedback is a
      FEEDBACK-1.

   3) If for small CIDs, and the size is larger than one octet, and the
      feedback starts with the two bits 11, the feedback starts with an
      Add-CID octet.  If the size is 2, it is followed by FEEDBACK-1.
      If the size is larger than 2, the Add-CID is followed by
      FEEDBACK-2.

   4) Otherwise, there is no Add-CID octet, and the feedback starts with
      a FEEDBACK-2.

5.2.3.  ROHC IR packet type

   The IR header associates a CID with a profile, and typically also
   initializes the context.  It can typically also refresh (parts of)
   the context.  It has the following general format.

```
     0   1   2   3   4   5   6   7
    --- --- --- --- --- --- --- ---
   :          Add-CID octet        : if for small CIDs and (CID != 0)
   +---+---+---+---+---+---+---+---+
   | 1   1   1   1   1   1   0 | x | IR type octet
   +---+---+---+---+---+---+---+---+
   :                               :
   /       0-2 octets of CID       / 1-2 octets if for large CIDs
   :                               :
   +---+---+---+---+---+---+---+---+
   |            Profile            | 1 octet
   +---+---+---+---+---+---+---+---+
   |             CRC               | 1 octet
   +---+---+---+---+---+---+---+---+
   |                               |
   / profile specific information  / variable length
   |                               |
   +---+---+---+---+---+---+---+---+
```
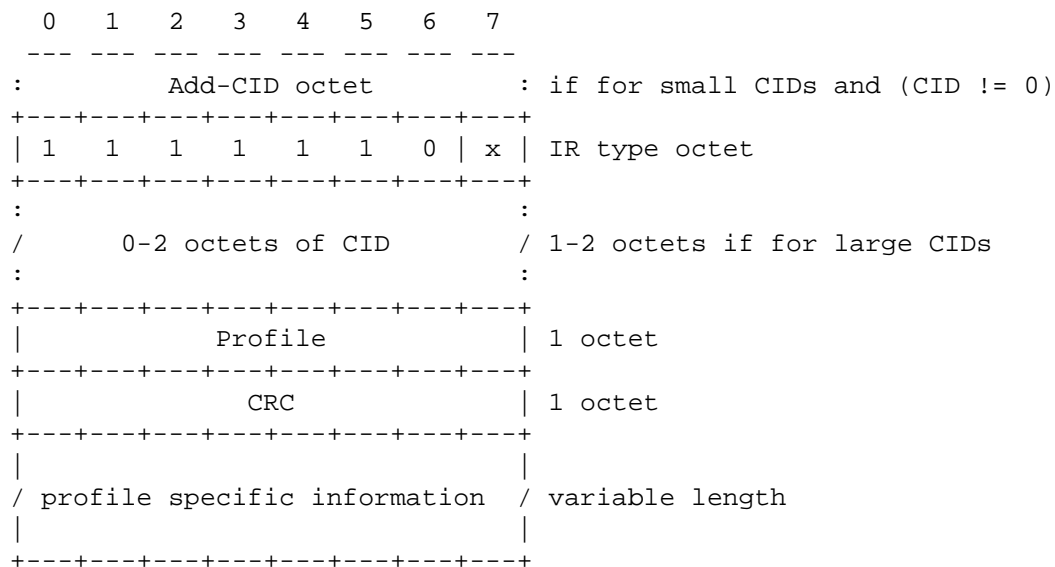
     x:  Profile specific information.  Interpreted according to the
         profile indicated in the Profile field.

   Profile: The profile to be associated with the CID.  In the IR
      packet, the profile identifier is abbreviated to the 8 least
      significant bits.  It selects the highest-number profile in the
      channel state parameter PROFILES that matches the 8 LSBs given.

   CRC: 8-bit CRC computed using the polynomial of section 5.9.1.  Its
      coverage is profile-dependent, but it MUST cover at least the
      initial part of the packet ending with the Profile field.  Any
      information which initializes the context of the decompressor
      should be protected by the CRC.

   Profile specific information: The contents of this part of the IR
      packet are defined by the individual profiles.  Interpreted
      according to the profile indicated in the Profile field.

5.2.4.  ROHC IR-DYN packet type

   In contrast to the IR header, the IR-DYN header can never initialize
   an uninitialized context.  However, it can redefine what profile is
   associated with a context, see for example 5.11 (ROHC UDP) and 5.12
   (ROHC ESP).  Thus the type needs to be reserved at the framework
   level.  The IR-DYN header typically also initializes or refreshes
   parts of a context, typically the dynamic part.  It has the following
   general format:

```
   0   1   2   3   4   5   6   7
 --- --- --- --- --- --- --- ---
:          Add-CID octet         : if for small CIDs and (CID != 0)
+---+---+---+---+---+---+---+---+
| 1   1   1   1   1   0   0   0 | IR-DYN type octet
+---+---+---+---+---+---+---+---+
:                               :
/      0-2 octets of CID        / 1-2 octets if for large CIDs
:                               :
+---+---+---+---+---+---+---+---+
|            Profile            | 1 octet
+---+---+---+---+---+---+---+---+
|             CRC               | 1 octet
+---+---+---+---+---+---+---+---+
|                               |
/ profile specific information  / variable length
|                               |
+---+---+---+---+---+---+---+---+
```

      Profile: The profile to be associated with the CID.  This is
          abbreviated in the same way as with IR packets.

CRC: 8-bit CRC computed using the polynomial of section 5.9.1.
    Its coverage is profile-dependent, but it MUST cover at least
    the initial part of the packet ending with the Profile field.
    Any information which initializes the context of the
    decompressor should be protected by the CRC.

Profile specific information: This part of the IR packet is
    defined by individual profiles.  It is interpreted according
    to the profile indicated in the Profile field.

5.2.5.  ROHC segmentation

   Some link layers may provide a much more efficient service if the set
   of different packet sizes to be transported is kept small.  For such
   link layers, these sizes will normally be chosen to transport
   frequently occurring packets efficiently, with less frequently
   occurring packets possibly adapted to the next larger size by the
   addition of padding.  The link layer may, however, be limited in the
   size of packets it can offer in this efficient mode, or it may be
   desirable to request only a limited largest size.  To accommodate the
   occasional packet that is larger than that largest size negotiated,
   ROHC defines a simple segmentation protocol.

5.2.5.1.  Segmentation usage considerations

   The segmentation protocol defined in ROHC is not particularly
   efficient.  It is not intended to replace link layer segmentation
   functions; these SHOULD be used whenever available and efficient for
   the task at hand.

   ROHC segmentation should only be used for occasional packets with
   sizes larger than what is efficient to accommodate, e.g., due to
   exceptionally large ROHC headers.  The segmentation scheme was
   designed to reduce packet size variations that may occur due to
   outliers in the header size distribution.  In other cases,
   segmentation should be done at lower layers.  The segmentation scheme
   should only be used for packet sizes that are larger than the maximum
   size in the allowed set of sizes from the lower layers.

   In summary, ROHC segmentation should be used with a relatively low
   frequency in the packet flow.  If this cannot be ensured,
   segmentation should be performed at lower layers.

5.2.5.2.  Segmentation protocol

   Segment Packet

      0   1   2   3   4   5   6   7
    +---+---+---+---+---+---+---+---+
    | 1   1   1   1   1   1   1 | F |
    +---+---+---+---+---+---+---+---+
    /           Segment            /  variable length
    +---+---+---+---+---+---+---+---+
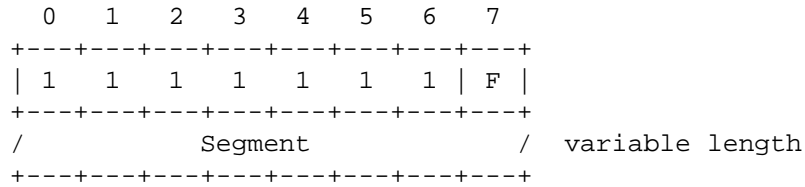
   F: Final bit.  If set, it indicates that this is the last segment of
   a reconstructed unit.

   The segment header may be preceded by padding octets and/or feedback.
   It never carries a CID.

   All segment header packets for one reconstructed unit have to be sent
   consecutively on a channel, i.e., any non-segment-header packet
   following a nonfinal segment header aborts the reassembly of the
   current reconstructed unit and causes the decompressor to discard the
   nonfinal segments received on this channel so far.  When a final
   segment header is received, the decompressor reassembles the segment
   carried in this packet and any nonfinal segments that immediately
   preceded it into a single reconstructed unit, in the order they were
   received.  The reconstructed unit has the format:

   Reconstructed Unit

      0   1   2   3   4   5   6   7
    +---+---+---+---+---+---+---+---+
    |                              |
    /   Reconstructed ROHC packet  /  variable length
    |                              |
    +---+---+---+---+---+---+---+---+
    /             CRC              /  4 octets
    +---+---+---+---+---+---+---+---+
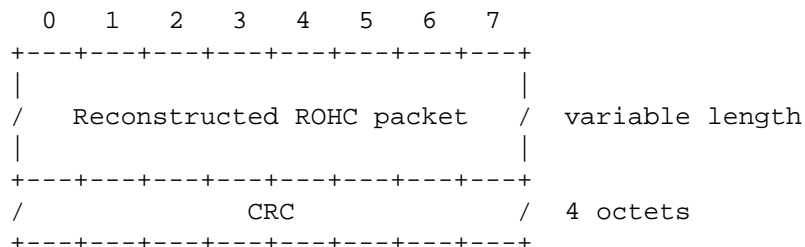
   The CRC is used by the decompressor to validate the reconstructed
   unit.  It uses the FCS-32 algorithm with the following generator
   polynomial: $x^0 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$ [HDLC].  If the reconstructed
   unit is 4 octets or less, or if the CRC fails, or if it is larger
   than the channel parameter MRRU (see 5.1.1), the reconstructed unit
   MUST be discarded by the decompressor.

If the CRC succeeds, the reconstructed ROHC packet is interpreted as
a ROHC Header, optionally followed by a payload.  Note that this
means that there can be no padding and no feedback in the
reconstructed unit, and that the CID is derived from the initial
octets of the reconstructed unit.

(It should be noted that the ROHC segmentation protocol was inspired
by SEAL by Steve Deering et al., which later became ATM AAL5.  The
same arguments for not having sequence numbers in the segments but
instead providing a strong CRC in the reconstructed unit apply here
as well.  Note that, as a result of this protocol, there is no way in
ROHC to make any use of a segment that has residual bit errors.)

5.2.6.  ROHC initial decompressor processing

The following packet types are reserved at the framework level in the
ROHC scheme:

```
1110:     Padding or Add-CID octet
11110:    Feedback
11111000: IR-DYN packet
1111110:  IR packet
1111111:  Segment
```

Other packet types can be used at will by individual profiles.

The following steps is an outline of initial decompressor processing
which upon reception of a ROHC packet can determine its contents.

1) If the first octet is a Padding Octet (11100000),
   strip away all initial Padding Octets and goto next step.

2) If the first remaining octet starts with 1110, it is an Add-CID
   octet:

       remember the Add-CID octet; remove the octet.

3) If the first remaining octet starts with 11110, and an Add-CID
   octet was found in step 2),

       an error has occurred; the header MUST be discarded without
       further action.

4) If the first remaining octet starts with 11110, and an Add-CID
   octet was not found in step 2), this is feedback:

       find the size of the feedback data, call it s;
       remove the feedback type octet;

        remove the Size octet if Code is 0;
        send feedback data of length s to the same-side associated
        compressor;
        if packet exhausted, stop; otherwise goto 2).

   5) If the first remaining octet starts with 1111111, this is a
      segment:

        attempt reconstruction using the segmentation protocol
        (5.2.5).  If a reconstructed packet is not produced, this
        finishes the processing of the original packet.  If a
        reconstructed packet is produced, it is fed into step 1)
        above.  Padding, segments, and feedback are not allowed in
        reconstructed packets, so when processing them, steps 1),
        4), and 5) are modified so that the packet is discarded
        without further action when their conditions match.

   6) Here, it is known that the rest is forward information (unless the
      header is damaged).

   7) If the forward traffic uses small CIDs, there is no large CID in
      the packet.  If an Add-CID immediately preceded the packet type
      (step 2), it has the CID of the Add-CID; otherwise it has CID 0.

   8) If the forward traffic uses large CIDs, the CID starts with the
      second remaining octet.  If the first bit(s) of that octet are not
      0 or 10, the packet MUST be discarded without further action.  If
      an Add-CID octet immediately preceded the packet type (step 2),
      the packet MUST be discarded without further action.

   9) Use the CID to find the context.

   10) If the packet type is IR, the profile indicated in the IR packet
       determines how it is to be processed.  If the CRC fails to verify
       the packet, it MUST be discarded.  If a profile is indicated in
       the context, the logic of that profile determines what, if any,
       feedback is to be sent.  If no profile is noted in the context,
       no further action is taken.

   11) If the packet type is IR-DYN, the profile indicated in the IR-DYN
       packet determines how it is to be processed.

      a) If the CRC fails to verify the packet, it MUST be discarded.
         If a profile is indicated in the context, the logic of that
         profile determines what, if any, feedback is to be sent.  If no
         profile is noted in the context, no further action is taken.

   b) If the context has not been initialized by an IR packet, the
      packet MUST be discarded.  The logic of the profile indicated
      in the IR-DYN header (if verified by the CRC), determines what,
      if any, feedback is to be sent.

   12) Otherwise, the profile noted in the context determines how the
       rest of the packet is to be processed.  If the context has not
       been initialized by an IR packet, the packet MUST be discarded
       without further action.

   The procedure for finding the size of the feedback data is as
   follows:

   Examine the three bits which immediately follow the feedback packet
   type.  When these bits are
      1-7, the size of the feedback data is given by the bits;
      0,   a Size octet, which explicitly gives the size of the
           feedback data, is present after the feedback type octet.

5.2.7.  ROHC RTP packet formats from compressor to decompressor

   ROHC RTP uses three packet types to identify compressed headers, and
   two for initialization/refresh.  The format of a compressed packet
   can depend on the mode.  Therefore a naming scheme of the form

      <modes format is used in>-<packet type number>-<some property>

   is used to uniquely identify the format when necessary, e.g., UOR-2,
   R-1.  For exact formats of the packet types, see section 5.7.

   Packet type zero: R-0, R-0-CRC, UO-0.

      This, the minimal, packet type is used when parameters of all SN-
      functions are known by the decompressor, and the header to be
      compressed adheres to these functions.  Thus, only the W-LSB
      encoded RTP SN needs to be communicated.

      R-mode: Only if a CRC is present (packet type R-0-CRC) may the
      header be used as a reference for subsequent decompression.

      U-mode and O-mode: A small CRC is present in the UO-0 packet.

   Packet type 1: R-1, R-1-ID, R-1-TS, UO-1, UO-1-ID, UO-1-TS.

      This packet type is used when the number of bits needed for the SN
      exceeds those available in packet type zero, or when the
      parameters of the SN-functions for RTP TS or IP-ID change.

R-mode: R-1-* packets are not used as references for subsequent
decompression.  Values for other fields than the RTP TS or IP-ID
can be communicated using an extension, but they do not update the
context.

U-mode and O-mode: Only the values of RTP SN, RTP TS and IP-ID can
be used as references for future compression.  Nonupdating values
can be provided for other fields using an extension (UO-1-ID).

Packet type 2: UOR-2, UOR-2-ID, UOR-2-TS

This packet type can be used to change the parameters of any SN-
function, except those for most static fields.  Headers of packets
transferred using packet type 2 can be used as references for
subsequent decompression.

Packet type: IR

This packet type communicates the static part of the context,
i.e., the value of the constant SN-functions.  It can optionally
also communicate the dynamic part of the context, i.e., the
parameters of the nonconstant SN-functions.

Packet type: IR-DYN

This packet type communicates the dynamic part of the context,
i.e., the parameters of nonconstant SN-functions.

5.2.8.  Parameters needed for mode transition in ROHC RTP

The packet types IR (with dynamic information), IR-DYN, and UOR-2 are
common for all modes.  They can carry a mode parameter which can take
the values U = Unidirectional, O = Bidirectional Optimistic, and R =
Bidirectional Reliable.

Feedback of types ACK, NACK, and STATIC-NACK carry sequence numbers,
and feedback packets can also carry a mode parameter indicating the
desired compression mode: U, O, or R.

As a shorthand, the notation PACKET(mode) is used to indicate which
mode value a packet carries.  For example, an ACK with mode parameter
R is written ACK(R), and an UOR-2 with mode parameter O is written
UOR-2(O).

5.3.  Operation in Unidirectional mode

5.3.1.  Compressor states and logic (U-mode)

   Below is the state machine for the compressor in Unidirectional mode.
   Details of the transitions between states and compression logic are
   given subsequent to the figure.

```
                          Optimistic approach
        +------>------>------>------>------>------>------>------>------+
        |                                                             |
        |          Optimistic approach        Optimistic approach     |
        |         +------>------>------+      +------>------>------+    |
        |         |                    |      |                   |    |
        |         |                    v      |                   v    v
     +---------+            +----------+            +----------+
     | IR State |           | FO State |           | SO State |
     +---------+            +----------+            +----------+
        ^       ^                |      ^                |    |
        |       |    Timeout     |      |  Timeout / Update |  |
        |       +------<------<------+      +------<------<------+    |
        |                                                             |
        |                        Timeout                              |
        +------<------<------<------<------<------<------<------<------+
```
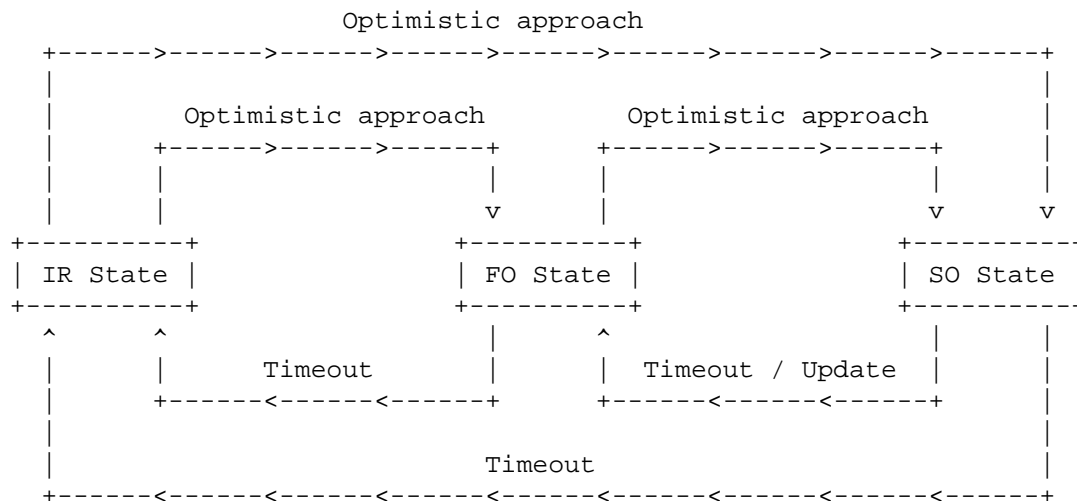
5.3.1.1.  State transition logic (U-mode)

   The transition logic for compression states in Unidirectional mode is
   based on three principles: the optimistic approach principle,
   timeouts, and the need for updates.

5.3.1.1.1.  Optimistic approach, upwards transition

   Transition to a higher compression state in Unidirectional mode is
   carried out according to the optimistic approach principle.  This
   means that the compressor transits to a higher compression state when
   it is fairly confident that the decompressor has received enough
   information to correctly decompress packets sent according to the
   higher compression state.

   When the compressor is in the IR state, it will stay there until it
   assumes that the decompressor has correctly received the static
   context information.  For transition from the FO to the SO state, the
   compressor should be confident that the decompressor has all
   parameters needed to decompress according to a fixed pattern.

The compressor normally obtains its confidence about decompressor
status by sending several packets with the same information according
to the lower compression state.  If the decompressor receives any of
these packets, it will be in sync with the compressor.  The number of
consecutive packets to send for confidence is not defined in this
document.

5.3.1.1.2.  Timeouts, downward transition

When the optimistic approach is taken as described above, there will
always be a possibility of failure since the decompressor may not
have received sufficient information for correct decompression.
Therefore, the compressor MUST periodically transit to lower
compression states.  Periodic transition to the IR state SHOULD be
carried out less often than transition to the FO state.  Two
different timeouts SHOULD therefore be used for these transitions.
For an example of how to implement periodic refreshes, see [IPHC]
chapters 3.3.1-3.3.2.

5.3.1.1.3.  Need for updates, downward transition

In addition to the downward state transitions carried out due to
periodic timeouts, the compressor must also immediately transit back
to the FO state when the header to be compressed does not conform to
the established pattern.

5.3.1.2.  Compression logic and packets used (U-mode)

The compressor chooses the smallest possible packet format that can
communicate the desired changes, and has the required number of bits
for W-LSB encoded values.

5.3.1.3.  Feedback in Unidirectional mode

The Unidirectional mode of operation is designed to operate over
links where a feedback channel is not available.  If a feedback
channel is available, however, the decompressor MAY send an
acknowledgment of successful decompression with the mode parameter
set to U (send an ACK(U)).  When the compressor receives such a
message, it MAY disable (or increase the interval between) periodic
IR refreshes.

5.3.2.  Decompressor states and logic (U-mode)

Below is the state machine for the decompressor in Unidirectional
mode.  Details of the transitions between states and decompression
logic are given subsequent to the figure.

```
                              Success
            +-->------>------>------>------>------>--+
            |                                        |
   No Static |            No Dynamic     Success    |    Success
    +-->--+  |             +-->--+    +--->----->---+  +-->--+
    |     |  |             |     |    |             |  |     |
    |     v  |             |     v    |             v  |     v
  +-------------+       +---------------+       +-------------+
  | No Context  |       | Static Context|       | Full Context|
  +-------------+       +---------------+       +-------------+
     ^                      |         ^                     |
     | k_2 out of n_2 failures |      | k_1 out of n_1 failures |
     +-----<------<------<-----+      +-----<------<------<-----+
```
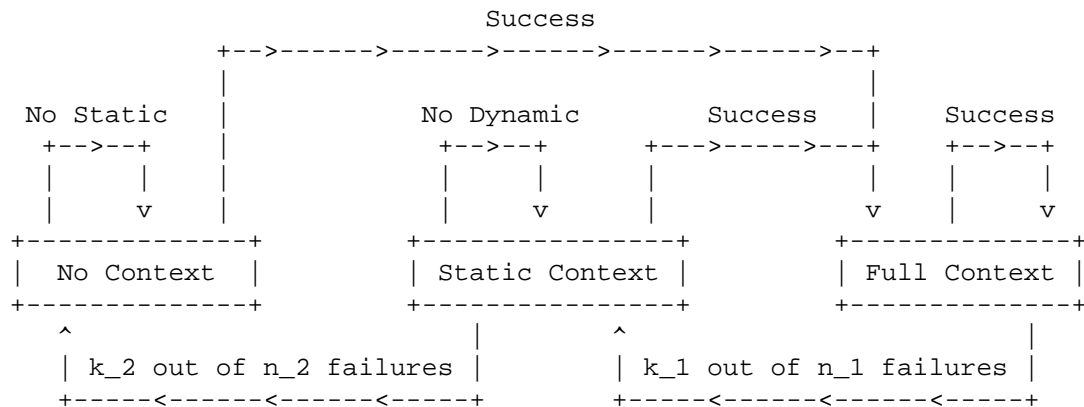
5.3.2.1.  State transition logic (U-mode)

   Successful decompression will always move the decompressor to the
   Full Context state.  Repeated failed decompression will force the
   decompressor to transit downwards to a lower state.  The decompressor
   does not attempt to decompress headers at all in the No Context and
   Static Context states unless sufficient information is included in
   the packet itself.

5.3.2.2.  Decompression logic (U-mode)

   Decompression in Unidirectional mode is carried out following three
   steps which are described in subsequent sections.

5.3.2.2.1.  Decide whether decompression is allowed

   In Full Context state, decompression may be attempted regardless of
   what kind of packet is received.  However, for the other states
   decompression is not always allowed.  In the No Context state only IR
   packets, which carry the static information fields, may be
   decompressed.  Further, when in the Static Context state, only
   packets carrying a 7- or 8-bit CRC can be decompressed (i.e., IR,
   IR-DYN, or UOR-2 packets).  If decompression may not be performed the
   packet is discarded, unless the optional delayed decompression
   mechanism is used, see section 6.1.

5.3.2.2.2.  Reconstruct and verify the header

   When reconstructing the header, the decompressor takes the header
   information already stored in the context and updates it with the
   information received in the current header.  (If the reconstructed
   header fails the CRC check, these updates MUST be undone.)

The sequence number is reconstructed by replacing the sequence number
LSBs in the context with those received in the header.  The resulting
value is then verified to be within the interpretation interval by
comparison with a previously reconstructed reference value v_ref (see
section 4.5.1).  If it is not within this interval, an adjustment is
applied by adding N x interval_size to the reconstructed value so
that the result is brought within the interpretation interval.  Note
that N can be negative.

If RTP Timestamp and IP Identification fields are not included in the
received header, they are supposed to be calculated from the sequence
number.  The IP Identifier usually increases by the same delta as the
sequence number and the timestamp by the same delta times a fixed
value.  See chapters 4.5.3 and 4.5.5 for details about how these
fields are encoded in compressed headers.

When working in Unidirectional mode, all compressed headers carry a
CRC which MUST be used to verify decompression.

5.3.2.2.3.  Actions upon CRC failure

This section is written so that it is applicable to all modes.

A mismatch in the CRC can be caused by one or more of:

1. residual bit errors in the current header

2. a damaged context due to residual bit errors in previous headers

3. many consecutive packets being lost between compressor and
   decompressor (this may cause the LSBs of the SN in compressed
   packets to be interpreted wrongly, because the decompressor has
   not moved the interpretation interval for lack of input -- in
   essence, a kind of context damage).

(Cases 2 and 3 do not apply to IR packets; case 3 does not apply to
IR-DYN packets.)  The 3-bit CRC present in some header formats will
eventually detect context damage reliably, since the probability of
undetected context damage decreases exponentially with each new
header processed.  However, residual bit errors in the current header
are only detected with good probability, not reliably.

When a CRC mismatch is caused by residual bit errors in the current
header (case 1 above), the decompressor should stay in its current
state to avoid unnecessary loss of subsequent packets.  On the other
hand, when the mismatch is caused by a damaged context (case 2), the
decompressor should attempt to repair the context locally.  If the
local repair attempt fails, it must move to a lower state to avoid

   delivering incorrect headers.  When the mismatch is caused by
   prolonged loss (case 3), the decompressor might attempt additional
   decompression attempts.  Note that case 3 does not occur in R-mode.

   The following actions MUST be taken when a CRC check fails:

   First, attempt to determine whether SN LSB wraparound (case 3) is
   likely, and if so, attempt a correction.  For this, the algorithm of
   section 5.3.2.2.4 MAY be used.  If another algorithm is used, it MUST
   have at least as high a rate of correct repairs as the one in
   5.3.2.2.4.  (This step is not applicable to R-mode.)

   Second, if the previous step did not attempt a correction, a repair
   should be attempted under the assumption that the reference SN has
   been incorrectly updated.  For this, the algorithm of section
   5.3.2.2.5 MAY be used.  If another algorithm is used, it MUST have at
   least as high a rate of correct repairs as the one in 5.3.2.2.5.
   (This step is not applicable to R-mode.)

   If both the above steps fail, additional decompression attempts
   SHOULD NOT be made.  There are two possible reasons for the CRC
   failure: case 1 or unrecoverable context damage.  It is impossible to
   know for certain which of these is the actual cause.  The following
   rules are to be used:

   a. When CRC checks fail only occasionally, assume residual errors in
      the current header and simply discard the packet.  NACKs SHOULD
      NOT be sent at this time.

   b. In the Full Context state: When the CRC check of k_1 out of the
      last n_1 decompressed packets have failed, context damage SHOULD
      be assumed and a NACK SHOULD be sent in O- and R-mode.  The
      decompressor moves to the Static Context state and discards all
      packets until an update (IR, IR-DYN, UOR-2) which passes the CRC
      check is received.

   c. In the Static Context state: When the CRC check of k_2 out of the
      last n_2 updates (IR, IR-DYN, UOR-2) have failed, static context
      damage SHOULD be assumed and a STATIC-NACK is sent in O- and R-
      mode.  The decompressor moves to the No Context state.

   d. In the No Context state: The decompressor discards all packets
      until a static update (IR) which passes the CRC check is received.
      (In O-mode and R-mode, feedback is sent according to sections
      5.4.2.2 and 5.5.2.2, respectively.)

Note that appropriate values for k_1, n_1, k_2, and n_2, are related
to the residual error rate of the link.  When the residual error rate
is close to zero, k_1 = n_1 = k_2 = n_2 = 1 may be appropriate.

5.3.2.2.4.  Correction of SN LSB wraparound

When many consecutive packets are lost there will be a risk of
sequence number LSB wraparound, i.e., the SN LSBs being interpreted
wrongly because the interpretation interval has not moved for lack of
input.  The decompressor might be able to detect this situation and
avoid context damage by using a local clock.  The following algorithm
MAY be used:

a. The decompressor notes the arrival time, a(i), of each incoming
   packet i.  Arrival times of packets where decompression fails are
   discarded.

b. When decompression fails, the decompressor computes INTERVAL =
   a(i) - a(i - 1), i.e., the time elapsed between the arrival of the
   previous, correctly decompressed packet and the current packet.

c. If wraparound has occurred, INTERVAL will correspond to at least
   2^k inter-packet times, where k is the number of SN bits in the
   current header.  On the basis of an estimate of the packet inter-
   arrival time, obtained for example using a moving average of
   arrival times, TS_STRIDE, or TS_TIME, the decompressor judges if
   INTERVAL can correspond to 2^k inter-packet times.

d. If INTERVAL is judged to be at least 2^k packet inter-arrival
   times, the decompressor adds 2^k to the reference SN and attempts
   to decompress the packet using the new reference SN.

e. If this decompression succeeds, the decompressor updates the
   context but SHOULD NOT deliver the packet to upper layers.  The
   following packet is also decompressed and updates the context if
   its CRC succeeds, but SHOULD be discarded.  If decompression of
   the third packet using the new context also succeeds, the context
   repair is deemed successful and this and subsequent decompressed
   packets are delivered to the upper layers.

f. If any of the three decompression attempts in d. and e. fails, the
   decompressor discards the packets and acts according to rules a)
   through c) of section 5.3.2.2.3.

Using this mechanism, the decompressor may be able to repair the
context after excessive loss, at the expense of discarding two
packets.

5.3.2.2.5.  Repair of incorrect SN updates

   The CRC can fail to detect residual errors in the compressed header
   because of its limited length, i.e., the incorrectly decompressed
   packet can happen to have the same CRC as the original uncompressed
   packet.  The incorrect decompressed header will then update the
   context.  This can lead to an erroneous reference SN being used in
   W-LSB decoding, as the reference SN is updated for each successfully
   decompressed header of certain types.

   In this situation, the decompressor will detect the incorrect
   decompression of the following packet with high probability, but it
   does not know the reason for the failure.  The following mechanism
   allows the decompressor to judge if the context was updated
   incorrectly by an earlier packet and, if so, to attempt a repair.

   a. The decompressor maintains two decompressed sequence numbers: the
      last one (ref 0) and the one before that (ref -1).

   b. When receiving a compressed header the SN (SN curr1) is
      decompressed using ref 0 as the reference.  The other header
      fields are decompressed using this decompressed SN curr1.  (This
      is part of the normal decompression procedure prior to any CRC
      test failures.)

   c. If the decompressed header generated in b. passes the CRC test,
      the references are shifted as follows:

           ref -1 = ref 0
           ref  0 = SN curr1.

   d. If the header generated in b. does not pass the CRC test, and the
      SN (SN curr2) generated when using ref -1 as the reference is
      different from SN curr1, an additional decompression attempt is
      performed based on SN curr2 as the decompressed SN.

   e. If the decompressed header generated in b. does not pass the CRC
      test and SN curr2 is the same as SN curr1, an additional
      decompression attempt is not useful and is not attempted.

   f. If the decompressed header generated in d. passes the CRC test,
      ref -1 is not changed while ref 0 is set to SN curr2.

   g. If the decompressed header generated in d. does not pass the CRC
      test, the decompressor acts according to rules a) through c) of
      section 5.3.2.2.3.

   The purpose of this algorithm is to repair the context.  If the
   header generated in d. passes the CRC test, the references are
   updated according to f., but two more headers MUST also be
   successfully decompressed before the repair is deemed successful.  Of
   the three successful headers, the first two SHOULD be discarded and
   only the third delivered to upper layers.  If decompression of any of
   the three headers fails, the decompressor MUST discard that header
   and the previously generated headers, and act according to rules a)
   through c) of section 5.3.2.2.3.

5.3.2.3.  Feedback in Unidirectional mode

   To improve performance for the Unidirectional mode over a link that
   does have a feedback channel, the decompressor MAY send an
   acknowledgment when decompression succeeds.  Setting the mode
   parameter in the ACK packet to U indicates that the compressor is to
   stay in Unidirectional mode.  When receiving an ACK(U), the
   compressor should reduce the frequency of IR packets since the static
   information has been correctly received, but it is not required to
   stop sending IR packets.  If IR packets continue to arrive, the
   decompressor MAY repeat the ACK(U), but it SHOULD NOT repeat the
   ACK(U) continuously.

5.4.  Operation in Bidirectional Optimistic mode

5.4.1.  Compressor states and logic (O-mode)

   Below is the state machine for the compressor in Bidirectional
   Optimistic mode.  The details of each state, state transitions, and
   compression logic are given subsequent to the figure.

                           Optimistic approach / ACK
      +------>------>------>------>------>------>------>------>------+
      |                                                             |
      |        Optimistic appr. / ACK     Optimistic appr. /ACK   ACK |
      |        +------>------>------+      +------>--- -->-----+  +->--+
      |        |                   |      |                   |  |    |
      |        |                   v      |                   v  |    v
   +---------+             +----------+             +----------+
   | IR State |            | FO State |             | SO State |
   +---------+             +----------+             +----------+
     ^      ^                  |      ^                  |    |
     |      |   STATIC-NACK     |      |   NACK / Update  |    |
     |      +------<------<------+      +------<------<------+    |
     |                                                           |
     |                       STATIC-NACK                         |
     +------<------<------<------<------<------<------<------<------+
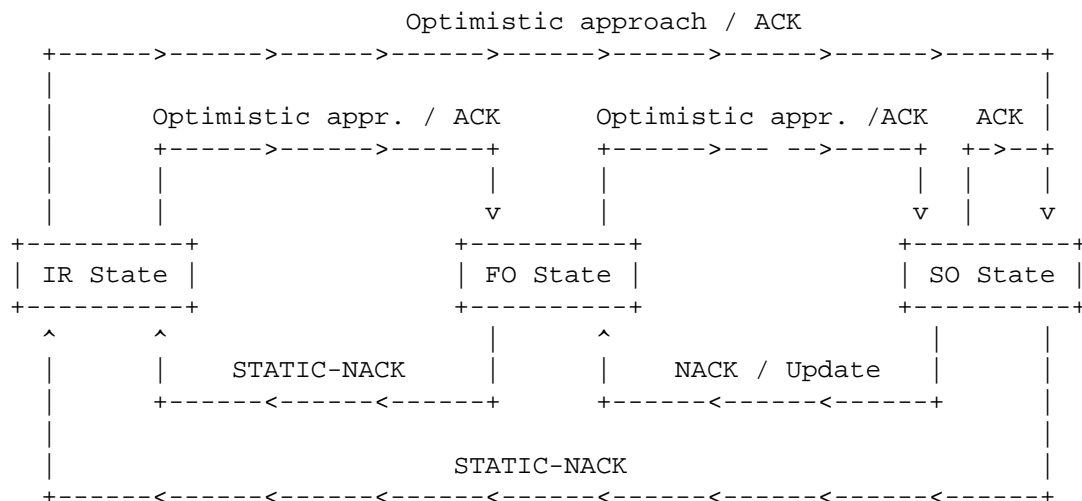
5.4.1.1.  State transition logic

   The transition logic for compression states in Bidirectional
   Optimistic mode has much in common with the logic of the
   Unidirectional mode.  The optimistic approach principle and
   transitions occasioned by the need for updates work in the same way
   as described in chapter 5.3.1.  However, in Optimistic mode there are
   no timeouts.  Instead, the Optimistic mode makes use of feedback from
   decompressor to compressor for transitions in the backward direction
   and for OPTIONAL improved forward transition.

5.4.1.1.1.  Negative acknowledgments (NACKs), downward transition

   Negative acknowledgments (NACKs), also called context requests,
   obviate the periodic updates needed in Unidirectional mode.  Upon
   reception of a NACK the compressor transits back to the FO state and
   sends updates (IR-DYN, UOR-2, or possibly IR) to the decompressor.
   NACKs carry the SN of the latest packet successfully decompressed,
   and this information MAY be used by the compressor to determine what
   fields need to be updated.

   Similarly, reception of a STATIC-NACK packet makes the compressor
   transit back to the IR state.

5.4.1.1.2.  Optional acknowledgments, upwards transition

   In addition to NACKs, positive feedback (ACKs) MAY also be used for
   UOR-2 packets in the Bidirectional Optimistic mode.  Upon reception
   of an ACK for an updating packet, the compressor knows that the
   decompressor has received the acknowledged packet and the transition
   to a higher compression state can be carried out immediately.  This
   functionality is optional, so a compressor MUST NOT expect to get
   such ACKs initially.

   The compressor MAY use the following algorithm to determine when to
   expect ACKs for UOR-2 packets.  Let an update event be when a
   sequence of UOR-2 headers are sent to communicate an irregularity in
   the packet stream.  When ACKs have been received for $k_3$ out of the
   last $n_3$ update events, the compressor will expect ACKs.  A
   compressor which expects ACKs will repeat updates (possibly not in
   every packet) until an ACK is received.

5.4.1.2.  Compression logic and packets used

   The compression logic is the same for the Bidirectional Optimistic
   mode as for the Unidirectional mode (see section 5.3.1.2).

5.4.2.  Decompressor states and logic (O-mode)

   The decompression states and the state transition logic are the same
   as for the Unidirectional case (see section 5.3.2).  What differs is
   the decompression and feedback logic.

5.4.2.1.  Decompression logic, timer-based timestamp decompression

   In Bidirectional mode (or if there is some other way for the
   compressor to obtain the decompressor's clock resolution and the
   link's jitter), timer-based timestamp decompression may be used to
   improve compression efficiency when RTP Timestamp values are
   proportional to wall-clock time.  The mechanisms used are those
   described in 4.5.4.

5.4.2.2.  Feedback logic (O-mode)

   The feedback logic defines what feedback to send due to different
   events when operating in the various states.  As mentioned above,
   there are three principal kinds of feedback; ACK, NACK and STATIC-
   NACK.  Further, the logic described below will refer to different
   kinds of packets that can be received by the decompressor;
   Initialization and Refresh (IR) packets, IR packets without static
   information (IR-DYN) and type 2 packets (UOR-2), or type 1 (UO-1) and
   type 0 packets (UO-0).  A type 0 packet carries a packet header
   compressed according to a fixed pattern, while type 1, 2 and IR-DYN
   packets are used when this pattern is broken.

   Below, rules are defined stating which feedback to use when.  If the
   optional feedback is used once, the decompressor is REQUIRED to
   continue to send optional feedback for the lifetime of the packet
   stream.

   State Actions

   NC:  - When an IR packet passes the CRC check, send an ACK(O).
        - When receiving a type 0, 1, 2 or IR-DYN packet, or an IR
          packet has failed the CRC check, send a STATIC-NACK(O),
          subject to the considerations at the beginning of section
          5.7.6.

   SC:  - When an IR packet is correctly decompressed, send an ACK(O).
        - When a type 2 or an IR-DYN packet is correctly decompressed,
          optionally send an ACK(O).
        - When a type 0 or 1 packet is received, treat it as a
          mismatching CRC and use the logic of section 5.3.2.2.3 to
          decide if a NACK(O) should be sent.

           - When decompression of a type 2 packet, an IR-DYN packet or an
             IR packet has failed, use the logic of section 5.3.2.2.3 to
             decide if a STATIC-NACK(O) should be sent.

       FC:  - When an IR packet is correctly decompressed, send an ACK(O).
            - When a type 2 or an IR-DYN packet is correctly decompressed,
              optionally send an ACK(O).
            - When a type 0 or 1 packet is correctly decompressed, no
              feedback is sent.
            - When any packet fails the CRC check, use the logic of
              5.3.2.2.3 to decide if a NACK(O) should be sent.

5.5.  Operation in Bidirectional Reliable mode

5.5.1.  Compressor states and logic (R-mode)

   Below is the state machine for the compressor in Bidirectional
   Reliable mode.  The details of each state, state transitions, and
   compression logic are given subsequent to the figure.

```
                                    ACK
        +------>------>------>------>------>------>------>------+
        |                                                      |
        |            ACK                      ACK              | ACK
        |     +------>------>------+    +------>------>------+  +->-+
        |     |      |             |    |      |            |  |   |
        |     |      |             v    |      |            v  |   v
    +---------+      |         +---------+     |        +----------+
    | IR State |     |         | FO State |    |        | SO State |
    +---------+      |         +---------+     |        +----------+
       ^       ^              |        ^             |        |
       |       |  STATIC-NACK |        |  NACK / Update |     |
       |       +------<------<------+   +------<------<------+ |
       |                                                      |
       |                      STATIC-NACK                     |
       +------<------<------<------<------<------<------<------+
```
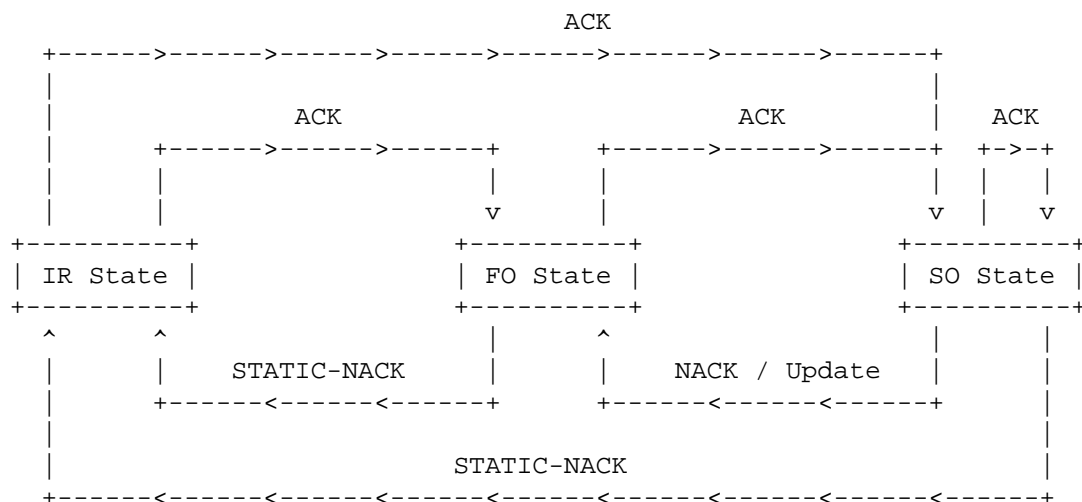
5.5.1.1.  State transition logic (R-mode)

   The transition logic for compression states in Reliable mode is based
   on three principles: the secure reference principle, the need for
   updates, and negative acknowledgments.

5.5.1.1.1.  Upwards transition

   The upwards transition is determined by the secure reference
   principle.  The transition procedure is similar to the one described
   in section 5.3.1.1.1, with one important difference: the compressor

bases its confidence only on acknowledgments received from the
decompressor.  This ensures that the synchronization between the
compression context and decompression context will never be lost due
to packet losses.

5.5.1.1.2.  Downward transition

Downward transitions are triggered by the need for updates or by
negative acknowledgment (NACKs and STATIC_NACKs), as described in
section 5.3.1.1.3 and 5.4.1.1.1, respectively.  Note that NACKs
should rarely occur in R-mode because of the secure reference used
(see fourth paragraph of next section).

5.5.1.2.  Compression logic and packets used (R-mode)

The compressor starts in the IR state by sending IR packets.  It
transits to the FO state once it receives a valid ACK for an IR
packet sent (an ACK can only be valid if it refers to an SN sent
earlier).  In the FO state, it sends the smallest packets that can
communicate the changes, according to W-LSB or other encoding rules.
Those packets could be of type R-1*, UOR-2, or even IR-DYN.

The compressor will transit to the SO state after it has determined
the presence of a string (see section 2), while also being confident
that the decompressor has the string parameters.  The confidence can
be based on ACKs.  For example, in a typical case where the string
pattern has the form of non-SN-field = SN * slope + offset, one ACK
is enough if the slope has been previously established by the
decompressor (i.e., only the new offset needs to be synchronized).
Otherwise, two ACKs are required since the decompressor needs two
headers to learn both the new slope and the new offset.  In the SO
state, R-0* packets will be sent.

Note that a direct transition from the IR state to the SO state is
possible.

The secure reference principle is enforced in both compression and
decompression logic.  The principle means that only a packet carrying
a 7- or 8-bit CRC can update the decompression context and be used as
a reference for subsequent decompression.  Consequently, only field
values of update packets need to be added to the encoding sliding
windows (see 4.5) maintained by the compressor.

Reasons for the compressor to send update packets include:

1) The update may lead to a transition to higher compression
   efficiency (meaning either a higher compression state or smaller
   packets in the same state).

   2) It is desirable to shrink sliding windows.  Windows are only
      shrunk when an ACK is received.

      The generation of a CRC is infrequent since it is only needed for
      an update packet.

   One algorithm for sending update packets could be:

     * Let pRTT be the number of packets that are sent during one
       round-trip time.  In the SO state, when (64 - pRTT) headers have
       been sent since the last acked reference, the compressor will
       send m1 consecutive R-0-CRC headers, then send (pRTT - m1) R-0
       headers.  After these headers have been sent, if the compressor
       has not received an ACK to at least one of the previously sent
       R0-CRC, it sends R-0-CRC headers continuously until it receives a
       corresponding ACK.  m1 is an implementation parameter, which can
       be as large as pRTT.

     * In the FO state, m2 UOR-2 headers are sent when there is a
       pattern change, after which the compressor sends (pRTT - m2)
       R-1-* headers.  m2 is an implementation parameter, which can be
       as large as pRTT.  At that time, if the compressor has not
       received enough ACKs to the previously sent UOR-2 packets in
       order to transit to SO state, it can repeat the cycle with the
       same m2, or repeat the cycle with a larger m2, or send UOR-2
       headers continuously (m2 = pRTT).  The operation stops when the
       compressor has received enough ACKs to make the transition.

   An algorithm for processing ACKs could be:

     * Upon reception of an ACK, the compressor first derives the
       complete SN (see section 5.7.6.1).  Then it searches the sliding
       window for an update packet that has the same SN.  If found, that
       packet is the one being ACKed.  Otherwise, the ACK is invalid and
       MUST be discarded.

     * It is possible, although unlikely, that residual errors on the
       reverse channel could cause a packet to mimic a valid ACK
       feedback.  The compressor may use a local clock to reduce the
       probability of processing such a mistaken ACK.  After finding the
       update packet as described above, the compressor can check the
       time elapsed since the packet was sent.  If the time is longer
       than RTT_U, or shorter than RTT_L, the compressor may choose to
       discard the ACK.  RTT_U and RTT_L correspond to an upper bound
       and lower bound of the RTT, respectively.  (These bounds should
       be chosen appropriately to allow some variation of RTT.)  Note
       that the only side effect of discarding a good ACK is slightly
       reduced compression efficiency.

5.5.2.  Decompressor states and logic (R-mode)

   The decompression states and the state transition logic are the same
   as for the Unidirectional case (see section 5.3.2).  What differs is
   the decompression and feedback logic.

5.5.2.1.  Decompression logic (R-mode)

   The rules for when decompression is allowed are the same as for U-
   mode.  Although the acking scheme in R-mode guarantees that non-
   decompressible packets are never sent by the compressor, residual
   errors can cause delivery of unexpected packets for which
   decompression should not be attempted.

   Decompression MUST follow the secure reference principle as described
   in 5.5.1.2.

   CRC verification is infrequent since only update packets carry CRCs.
   A CRC mismatch can only occur due to 1) residual bit errors in the
   current header, and/or 2) a damaged context due to residual bit
   errors in previous headers or feedback.  Although it is impossible to
   determine which is the actual cause, case 1 is more likely, as a
   previous header reconstructed according to a damaged packet is
   unlikely to pass the 7- or 8-bit CRC, and damaged packets are
   unlikely to result in feedback that damages the context.  The
   decompressor SHOULD act according to section 5.3.2.2.3 when CRCs
   fail, except that no local repair is performed.  Note that all the
   parameter numbers, k_1, n_1, k_2, and n_2, are applied to the update
   packets only (i.e., exclude R-0, R-1*).

5.5.2.2.  Feedback logic (R-mode)

   The feedback logic for the Bidirectional Reliable mode is as follows:

   - When an updating packet (i.e., a packet carrying a 7- or 8-bit CRC)
     is correctly decompressed, send an ACK(R), subject to the sparse
     ACK mechanism described below.

   - When context damage is detected, send a NACK(R) if in Full Context
     state, or a STATIC-NACK(R) if in Static Context state.

   - In No Context state, send a STATIC-NACK(R) when receiving a non-IR
     packet, subject to the considerations at the beginning of section
     5.7.6.  The decompressor SHOULD NOT send STATIC-NACK(R) when
     receiving an IR packet that fails the CRC check, as the compressor
     will stay in IR state and thus continue sending IR packets until a
     valid ACK is received (see section 5.5.1.2).

   - Feedback is never sent for packets not updating the context (i.e.,
     packets that do not carry a CRC)

   A mechanism called "Sparse ACK" can be applied to reduce the feedback
   overhead caused by a large RTT.  For a sequence of ACK-triggering
   events, a minimal set of ACKs MUST be sent:

   1) For a sequence of R-0-CRC packets, the first one MUST be ACKed.

   2) For a sequence of UOR-2, IR, or IR-DYN packets, the first N of
      them MUST be ACKEd, where N is the number of ACKs needed to give
      the compressor confidence that the decompressor has acquired the
      new string parameters (see second paragraph of 5.5.1.2).  In case
      the decompressor cannot determine the value of N, the default
      value 2 SHOULD be used.  If the subsequently received packets
      continue the same change pattern of header fields, sparse ACK can
      be applied.  Otherwise, each new pattern MUST be treated as a new
      sequence, i.e., the first N packets that exhibit a new pattern
      MUST be ACKed.

   After sending these minimal ACKs, the decompressor MAY choose to ACK
   only k subsequent packets per RTT ("Sparse ACKs"), where k is an
   implementation parameter.  To achieve robustness against loss of
   ACKs, k SHOULD be at least 1.

   To avoid ambiguity at the compressor, the decompressor MUST use the
   feedback format whose SN field length is equal to or larger than the
   one in the compressed packet that triggered the feedback.

   Context damage is detected according to the principles in 5.3.2.2.3.

   When the decompressor is capable of timer-based compression of the
   RTP Timestamp (e.g., it has access to a clock with sufficient
   resolution, and the jitter introduced internally in the receiving
   node is sufficiently small) it SHOULD signal that it is ready to do
   timer-based compression of the RTP Timestamp.  The compressor will
   then make a decision based on its knowledge of the channel and the
   observed properties of the packet stream.

5.6.  Mode transitions

   The decision to move from one compression mode to another is taken by
   the decompressor and the possible mode transitions are shown in the
   figure below.  Subsequent chapters describe how the transitions are
   performed together with exceptions for the compression and
   decompression functionality during transitions.

```
                    +-------------------------+
                    | Unidirectional (U) mode |
                    +-------------------------+
                     / ^                   \ ^
                    / / Feedback(U)         \ \ Feedback(U)
                   / /                       \ \
                  / /                         \ \
        Feedback(O) / /              Feedback(R) \ \
               v /                               v \
    +--------------------+   Feedback(R)   +------------------+
    | Optimistic (O) mode | --------------->| Reliable (R) mode |
    |                    | <--------------- |                  |
    +--------------------+   Feedback(O)   +------------------+
```
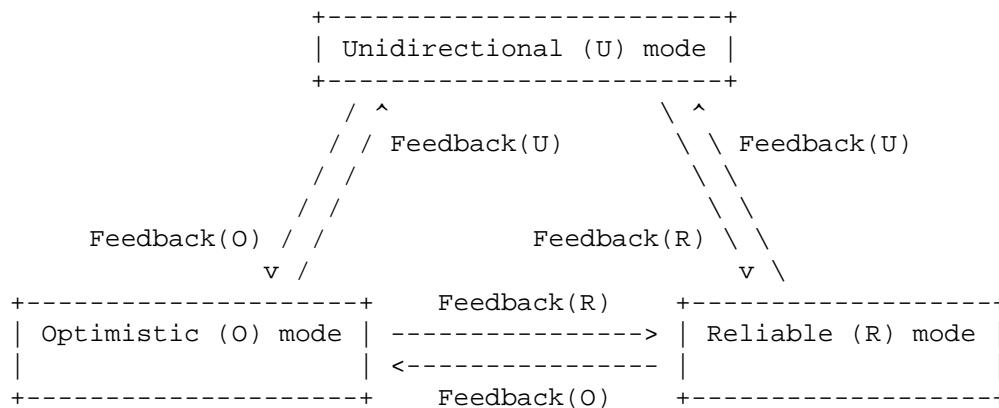
5.6.1.  Compression and decompression during mode transitions

   The following sections assume that, for each context, the compressor
   and decompressor maintain a variable whose value is the current
   compression mode for that context.  The value of the variable
   controls, for the context in question, which packet types to use,
   which actions to be taken, etc.

   As a safeguard against residual errors, all feedback sent during a
   mode transition MUST be protected by a CRC, i.e., the CRC option MUST
   be used.  A mode transition MUST NOT be initiated by feedback which
   is not protected by a CRC.

   The subsequent subsections define exactly when to change the value of
   the MODE variable.  When ROHC transits between compression modes,
   there are several cases where the behavior of compressor or
   decompressor must be restricted during the transition phase.  These
   restrictions are defined by exception parameters that specify which
   restrictions to apply.  The transition descriptions in subsequent
   chapters refer to these exception parameters and defines when they
   are set and to what values.  All mode related parameters are listed
   below together with their possible values, with explanations and
   restrictions:

   Parameters for the compressor side:

      - C_MODE:
         Possible values for the C_MODE parameter are (U)NIDIRECTIONAL,
         (O)PTIMISTIC and (R)ELIABLE.  C_MODE MUST be initialized to U.

      - C_TRANS:
         Possible values for the C_TRANS parameter are (P)ENDING and
         (D)ONE.  C_TRANS MUST be initialized to D.  When C_TRANS is P,
         it is REQUIRED

            1) that the compressor only use packet formats common to all
               modes,

            2) that mode information is included in packets sent, at least
               periodically,

            3) that the compressor not transit to the SO state,

            4) that new mode transition requests be ignored.

      Parameters for the decompressor side:

        - D_MODE:
           Possible values for the D_MODE parameter are (U)NIDIRECTIONAL,
           (O)PTIMISTIC and (R)ELIABLE.  D_MODE MUST be initialized to U.

        - D_TRANS:
           Possible values for the D_TRANS parameter are (I)NITIATED,
           (P)ENDING and (D)ONE.  D_TRANS MUST be initialized to D.  A
           mode transition can be initiated only when D_TRANS is D.  While
           D_TRANS is I, the decompressor sends a NACK or ACK carrying a
           CRC option for each received packet.

5.6.2.  Transition from Unidirectional to Optimistic mode

   When there is a feedback channel available, the decompressor may at
   any moment decide to initiate transition from Unidirectional to
   Bidirectional Optimistic mode.  Any feedback packet carrying a CRC
   can be used with the mode parameter set to O.  The decompressor can
   then directly start working in Optimistic mode.  The compressor
   transits from Unidirectional to Optimistic mode as soon as it
   receives any feedback packet that has the mode parameter set to O and
   that passes the CRC check.  The transition procedure is described
   below:

              Compressor                         Decompressor
            ---------------------------------------------
                  |                             |
                  |           ACK(O)/NACK(O) +-<-<-<-|  D_MODE = O
                  |              +-<-<-<-<-<-<-<-+       |
   C_MODE = O     |-<-<-<-+                            |
                  |                             |

   If the feedback packet is lost, the compressor will continue to work
   in Unidirectional mode, but as soon as any feedback packet reaches
   the compressor it will transit to Optimistic mode.

5.6.3.  From Optimistic to Reliable mode

   Transition from Optimistic to Reliable mode is permitted only after
   at least one packet has been correctly decompressed, which means that
   at least the static part of the context is established.  An ACK(R) or
   a NACK(R) feedback packet carrying a CRC is sent to initiate the mode
   transition.  The compressor MUST NOT use packet types 0 or 1 during
   transition.  The transition procedure is described below:

```
           Compressor                        Decompressor
           ---------------------------------------------
                   |                           |
                   |          ACK(R)/NACK(R) +-<-<-<-|  D_TRANS = I
                   |          +-<-<-<-<-<-<-+        |
C_TRANS = P        |-<-<-<-+                         |
C_MODE = R         |                                 |
                   |->->->-+ IR/IR-DYN/UOR-2(SN,R)   |
                   |       +->->->->->->->-+         |
                   |->-..                  +->->->-|  D_TRANS = P
                   |->-..                            |  D_MODE = R
                   |              ACK(SN,R)    +-<-<-<-|
                   |          +-<-<-<-<-<-<-+        |
C_TRANS = D        |-<-<-<-+                         |
                   |                                 |
                   |->->->-+   R-0*, R-1*             |
                   |       +->->->->->->->-+         |
                   |                       +->->->-|  D_TRANS = D
                   |                                 |
```

   As long as the decompressor has not received an UOR-2, IR-DYN, or IR
   packet with the mode transition parameter set to R, it must stay in
   Optimistic mode.  The compressor must not send packet types 1 or 0
   while C_TRANS is P, i.e., not until it has received an ACK for a
   UOR-2, IR-DYN, or IR packet sent with the mode transition parameter
   set to R.  When the decompressor receives packet types 0 or 1, after
   having ACKed an UOR-2, IR-DYN, or IR packet, it sets D_TRANS to D.

5.6.4.  From Unidirectional to Reliable mode

   The transition from Unidirectional to Reliable mode follows the same
   transition procedure as defined in section 5.6.3 above.

5.6.5.  From Reliable to Optimistic mode

   Either the ACK(O) or the NACK(O) feedback packet is used to initiate
   the transition from Reliable to Optimistic mode and the compressor
   MUST always run in the FO state during transition.  The transition
   procedure is described below:

```
              Compressor                    Decompressor
          ----------------------------------------------
              |                          |
              |            ACK(O)/NACK(O) +-<-<-<-|  D_TRANS = I
              |         +-<-<-<-<-<-<-<-+         |
 C_TRANS = P  |-<-<-<-+                           |
 C_MODE = O   |                                   |
              |->->->-+ IR/IR-DYN/UOR-2(SN,O)      |
              |       +->->->->->->->-+            |
              |->-..                  +->->->-|  D_MODE = O
              |->-..                              |
              |            ACK(SN,O)    +-<-<-<-|
              |         +-<-<-<-<-<-<-<-+         |
 C_TRANS = D  |-<-<-<-+                           |
              |                                   |
              |->->->-+  UO-0, UO-1*              |
              |       +->->->->->->->-+            |
              |                       +->->->-|  D_TRANS = D
              |                          |
```

   As long as the decompressor has not received an UOR-2, IR-DYN, or IR
   packet with the mode transition parameter set to O, it must stay in
   Reliable mode.  The compressor must not send packet types 0 or 1
   while C_TRANS is P, i.e., not until it has received an ACK for an
   UOR-2, IR-DYN, or IR packet sent with the mode transition parameter
   set to O.  When the decompressor receives packet types 0 or 1, after
   having ACKed the UOR-2, IR-DYN, or IR packet, it sets D_TRANS to D.

5.6.6.  Transition to Unidirectional mode

   The decompressor can force a transition back to Unidirectional mode
   if it desires to do so.  Regardless of which mode this transition
   starts from, a three-way handshake MUST be carried out to ensure
   correct transition on the compressor side.  The transition procedure
   is described below:

```
              Compressor                      Decompressor
           ------------------------------------------------
              |                             |
              |          ACK(U)/NACK(U) +-<-<-<-| D_TRANS = I
              |          +-<-<-<-<-<-<-<-+       |
  C_TRANS = P |-<-<-<-+                          |
  C_MODE = U  |                                  |
              |->->->-+ IR/IR-DYN/UOR-2(SN,U)    |
              |       +->->->->->->->-+          |
              |->-..                  +->->->-|
              |->-..                             |
              |          ACK(SN,U)    +-<-<-<-|
              |          +-<-<-<-<-<-<-<-+       |
  C_TRANS = D |-<-<-<-+                          |
              |                                  |
              |->->->-+  UO-0, UO-1*             |
              |       +->->->->->->->-+          |
              |                       +->->->-| D_TRANS = D, D_MODE= U
```

   After ACKing the first UOR-2(U), IR-DYN(U), or IR(U), the
   decompressor MUST continue to send feedback with the Mode parameter
   set to U until it receives packet types 0 or 1.

## 5.7.  Packet formats

   The following notation is used in this section:

      bits(X) = the number of bits for field X present in the compressed
               header (including extension).

      field(X) = the value of field X in the compressed header.

      context(X) = the value of field X as established in the context.

      value(X) = field(X) if X is present in the compressed header;
                = context(X) otherwise.

      hdr(X) = the value of field X in the uncompressed or
               decompressed header.

      Updating properties: Lists the fields in the context that are
         directly updated by processing the compressed header.  Note
         that there may be dependent fields that are implicitly also
         updated (e.g., an update to context(SN) often updates
         context(TS) as well).  See also section 5.2.7.

The following fields occur in several headers and extensions:

SN: The compressed RTP Sequence Number.

Compressed with W-LSB.  The interpretation intervals, see section
4.5.1, are defined as follows:

        p = 1                    if bits(SN) <= 4
        p = 2^(bits(SN)-5) - 1   if bits(SN) >  4

IP-ID: A compressed IP-ID field.

IP-ID fields in compressed base headers carry the compressed IP-ID
of the innermost IPv4 header whose corresponding RND flag is not
1.  The rules below assume that the IP-ID is for the innermost IP
header.  If it is for an outer IP header, the RND2 and NBO2 flags
are used instead of RND and NBO.

If value(RND) = 0, hdr(IP-ID) is compressed using Offset IP-ID
encoding (see section 4.5.5) using p = 0 and default-slope(IP-ID
offset) = 0.

If value(RND) = 1, IP-ID is the uncompressed hdr(IP-ID).  IP-ID is
then passed as additional octets at the end of the compressed
header, after any extensions.

If value(NBO) = 0, the octets of hdr(IP-ID) are swapped before
compression and after decompression.  The value of NBO is ignored
when value(RND) = 1.

TS: The compressed RTP Timestamp value.

If value(TIME_STRIDE) > 0, timer-based compression of the RTP
Timestamp is used (see section 4.5.4).

If value(Tsc) = 1, Scaled RTP Timestamp encoding is used before
compression (see section 4.5.3), and default-slope(TS) = 1.

If value(Tsc) = 0, the Timestamp value is compressed as-is, and
default-slope(TS) = value(TS_STRIDE).

The interpretation intervals, see section 4.5.1, are defined as
follows:

    p = 2^(bits(TS)-2) - 1

CRC: The CRC over the original, uncompressed, header.

   For 3-bit CRCs, the polynomial of section 5.9.2 is used.
   For 7-bit CRCs, the polynomial of section 5.9.2 is used.
   For 8-bit CRCs, the polynomial of section 5.9.1 is used.

M: RTP Marker bit.

   Context(M) is initially zero and is never updated.  value(M) = 1
   only when field(M) = 1.

The general format for a compressed RTP header is as follows:

```
   0   1   2   3   4   5   6   7
 --- --- --- --- --- --- --- ---
:          Add-CID octet         :  if for small CIDs and CID 1-15
+---+---+---+---+---+---+---+---+
|   first octet of base header  |  (with type indication)
+---+---+---+---+---+---+---+---+
:                               :
/   0, 1, or 2 octets of CID    /  1-2 octets if large CIDs
:                               :
+---+---+---+---+---+---+---+---+
/   remainder of base header    /  variable number of bits
+---+---+---+---+---+---+---+---+
:                               :
/     Extension (see 5.7.5)     /  extension, if X = 1 in base header
:                               :
 --- --- --- --- --- --- --- ---
:                               :
+   IP-ID of outer IPv4 header  +  2 octets, if value(RND2) = 1
:                               :
 --- --- --- --- --- --- --- ---
/    AH data for outer list     /  variable (see 5.8.4.2)
 --- --- --- --- --- --- --- ---
:                               :
+   GRE checksum (see 5.8.4.4)  +  2 octets, if GRE flag C = 1
:                               :
 --- --- --- --- --- --- --- ---
:                               :
+   IP-ID of inner IPv4 header  +  2 octets, if value(RND) = 1
:                               :
 --- --- --- --- --- --- --- ---
/    AH data for inner list     /  variable (see 5.8.4.2)
 --- --- --- --- --- --- --- ---
:                               :
+   GRE checksum (see 5.8.4.4)  +  2 octets, if GRE flag C = 1
:                               :
 --- --- --- --- --- --- --- ---
:                               :
+         UDP Checksum          +  2 octets,
:                               :  if context(UDP Checksum) != 0
 --- --- --- --- --- --- --- ---
```

Note that the order of the fields following the optional extension is
the same as the order between the fields in an uncompressed header.

In subsequent sections, the position of the large CID in the diagrams
is indicated using this notation:

```
+===+===+===+===+===+===+===+===+
```

Whether the UDP Checksum field is present or not is controlled by the value of the UDP Checksum in the context.  If nonzero, the UDP Checksum is enabled and sent along with each packet.  If zero, the UDP Checksum is disabled and not sent.  Should hdr(UDP Checksum) be nonzero when context(UDP Checksum) is zero, the header cannot be compressed.  It must be sent uncompressed or the context reinitialized using an IR packet.  Context(UDP Checksum) is updated only by IR or IR-DYN headers, never by UDP checksums sent in headers of type 2, 1, or 0.

When an IPv4 header is present in the static context, for which the corresponding RND flag has not been established to be 1, the packet types R-1 and UO-1 MUST NOT be used.

When no IPv4 header is present in the static context, or the RND flags for all IPv4 headers in the context have been established to be 1, the packet types R-1-ID, R-1-TS, UO-1-ID, and UO-1-TS MUST NOT be used.

While in the transient state in which an RND flag is being established, the packet types R-1-ID, R-1-TS, UO-1-ID, and UO-1-TS MUST NOT be used.  This implies that the RND flag(s) of the Extension 3 may have to be inspected before the format of a base header carrying an Extension 3 can be determined.

5.7.1. Packet type 0: UO-0, R-0, R-0-CRC

Packet type 0 is indicated by the first bit being 0:

R-0

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0   0 |          SN           |
+===+===+===+===+===+===+===+===+
```

   Updating properties: R-0 packets do not update any part of the context.

   R-0-CRC

     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 0   1 |            SN         |
   +===+===+===+===+===+===+===+===+
   |SN |            CRC            |
   +---+---+---+---+---+---+---+---+
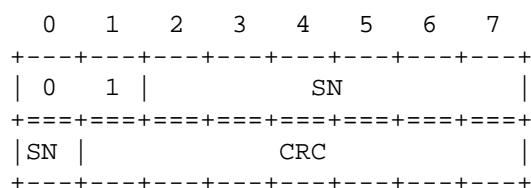
      Note: The SN field straddles the CID field.

      Updating properties: R-0-CRC packets update context(RTP Sequence
      Number).

   UO-0

     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 0 |       SN      |    CRC    |
   +===+===+===+===+===+===+===+===+
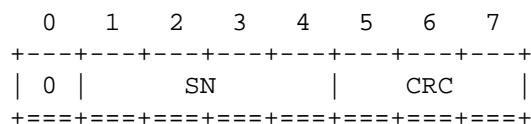
      Updating properties: UO-0 packets update the current value of
      context(RTP Sequence Number).

5.7.2. Packet type 1 (R-mode): R-1, R-1-TS, R-1-ID

   Packet type 1 is indicated by the first bits being 10:

   R-1
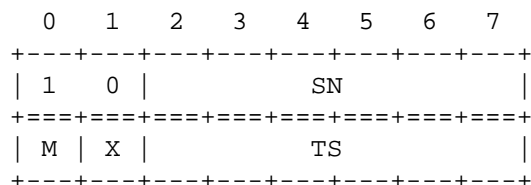
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   0 |            SN         |
   +===+===+===+===+===+===+===+===+
   | M | X |            TS         |
   +---+---+---+---+---+---+---+---+

      Note: R-1 cannot be used if the context contains at least one IPv4
      header with value(RND) = 0.  This disambiguates it from R-1-ID and
      R-1-TS.

R-1-ID

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   0 |           SN          |
+===+===+===+===+===+===+===+===+
| M | X |T=0|         IP-ID      |
+---+---+---+---+---+---+---+---+
```

   Note: R-1-ID cannot be used if there is no IPv4 header in the
   context or if value(RND) and value(RND2) are both 1.

R-1-TS

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   0 |           SN          |
+===+===+===+===+===+===+===+===+
| M | X |T=1|          TS        |
+---+---+---+---+---+---+---+---+
```

   Note: R-1-TS cannot be used if there is no IPv4 header in the
   context or if value(RND) and value(RND2) are both 1.

   X: X = 0 indicates that no extension is present;
      X = 1 indicates that an extension is present.

   T: T = 0 indicates format R-1-ID;
      T = 1 indicates format R-1-TS.

   Updating properties: R-1* headers do not update any part of the
   context.

5.7.3. Packet type 1 (U/O-mode): UO-1, UO-1-ID, UO-1-TS

   UO-1

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   0 |            TS         |
+===+===+===+===+===+===+===+===+
| M |      SN       |   CRC     |
+---+---+---+---+---+---+---+---+
```

   Note: UO-1 cannot be used if the context contains at least one
   IPv4 header with value(RND) = 0.  This disambiguates it from UO-
   1-ID and UO-1-TS.

UO-1-ID

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   0 |T=0|       IP-ID       |
+===+===+===+===+===+===+===+===+
| X |     SN        |   CRC     |
+---+---+---+---+---+---+---+---+
```
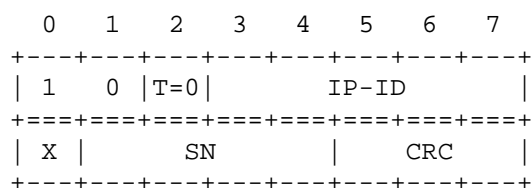
   Note: UO-1-ID cannot be used if there is no IPv4 header in the
   context or if value(RND) and value(RND2) are both 1.

UO-1-TS

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   0 |T=1|        TS         |
+===+===+===+===+===+===+===+===+
| M |     SN        |   CRC     |
+---+---+---+---+---+---+---+---+
```
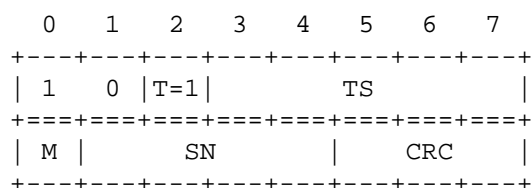
   Note: UO-1-TS cannot be used if there is no IPv4 header in the
   context or if value(RND) and value(RND2) are both 1.

   X: X = 0 indicates that no extension is present;
      X = 1 indicates that an extension is present.

   T: T = 0 indicates format UO-1-ID;
      T = 1 indicates format UO-1-TS.

   Updating properties: UO-1* packets update context(RTP Sequence
   Number).  UO-1 and UO-1-TS packets update context(RTP Timestamp).
   UO-1-ID packets update context(IP-ID).  Values provided in
   extensions, except those in other SN, TS, or IP-ID fields, do not
   update the context.

5.7.4. Packet type 2: UOR-2

   Packet type 2 is indicated by the first bits being 110:

   UOR-2

```
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   1   0 |        TS         |
   +===+===+===+===+===+===+===+===+
   |TS | M |            SN         |
   +---+---+---+---+---+---+---+---+
   | X |            CRC            |
   +---+---+---+---+---+---+---+---+
```

      Note: UOR-2 cannot be used if the context contains at least one
      IPv4 header with value(RND) = 0.  This disambiguates it from UOR-
      2-ID and UOR-2-TS.

      Note: The TS field straddles the CID field.

   UOR-2-ID

```
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   1   0 |       IP-ID       |
   +===+===+===+===+===+===+===+===+
   |T=0| M |            SN         |
   +---+---+---+---+---+---+---+---+
   | X |            CRC            |
   +---+---+---+---+---+---+---+---+
```

      Note: UOR-2-ID cannot be used if there is no IPv4 header in the
      context or if value(RND) and value(RND2) are both 1.

   UOR-2-TS

```
     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   1   0 |        TS         |
   +===+===+===+===+===+===+===+===+
   |T=1| M |            SN         |
   +---+---+---+---+---+---+---+---+
   | X |            CRC            |
   +---+---+---+---+---+---+---+---+
```

      Note: UOR-2-TS cannot be used if there is no IPv4 header in the
      context or if value(RND) and value(RND2) are both 1.

      X: X = 0 indicates that no extension is present;
         X = 1 indicates that an extension is present.

      T: T = 0 indicates format UOR-2-ID;
         T = 1 indicates format UOR-2-TS.


      Updating properties: All values provided in UOR-2* packets update
      the context, unless explicitly stated otherwise.

5.7.5.  Extension formats

   (Note: the term extension as used for additional information
   contained in the ROHC headers does not bear any relationship to the
   term extension header used in IP.)

   Fields in extensions are concatenated with the corresponding field in
   the base compressed header, if there is one.  Bits in an extension
   are less significant than bits in the base compressed header (see
   section 4.5.7).

   The TS field is scaled in all extensions, as it is in the base
   header, except optionally when using Extension 3 where the Tsc flag
   can indicate that the TS field is not scaled.  Value(TS_STRIDE) is
   used as the scale factor when scaling the TS field.

   In the following three extensions, the interpretation of the fields
   depends on whether there is a T-bit in the base compressed header,
   and if so, on the value of that field.  When there is no T-bit, +T
   and -T both mean TS.  This is the case when there are no IPv4 headers
   in the static context, and when all IPv4 headers in the static
   context have their corresponding RND flag set (i.e., RND = 1).

   If there is a T-bit,

      T = 1 indicates that +T is TS, and
                          -T is IP-ID;

      T = 0 indicates that +T is IP-ID, and
                          -T is TS.

   Extension 0:

        0   1   2   3   4   5   6   7
      +---+---+---+---+---+---+---+---+
      | 0   0 |    SN     |    +T     |
      +---+---+---+---+---+---+---+---+

Extension 1:

```
  +---+---+---+---+---+---+---+---+
  | 0   1 |    SN     |    +T     |
  +---+---+---+---+---+---+---+---+
  |               -T              |
  +---+---+---+---+---+---+---+---+
```

Extension 2:

```
  +---+---+---+---+---+---+---+---+
  | 1   0 |    SN     |    +T     |
  +---+---+---+---+---+---+---+---+
  |               +T              |
  +---+---+---+---+---+---+---+---+
  |               -T              |
  +---+---+---+---+---+---+---+---+
```

Extension 3 is a more elaborate extension which can give values for
fields other than SN, TS, and IP-ID.  Three optional flag octets
indicate changes to IP header(s) and RTP header, respectively.

Extension 3:

```
     0     1     2     3     4     5     6     7
  +-----+-----+-----+-----+-----+-----+-----+-----+
  | 1     1 |  S  |R-TS | Tsc |  I  | ip  | rtp |            (FLAGS)
  +-----+-----+-----+-----+-----+-----+-----+-----+
  |             Inner IP header flags    | ip2 |  if ip = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  |             Outer IP header flags           |  if ip2 = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  |                     SN                       |  if S = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  /       TS (encoded as in section 4.5.6)      /  1-4 octets,
   ..... ..... ..... ..... ..... ..... ..... .....  if R-TS = 1
  |                                              |
  /           Inner IP header fields             /  variable,
  |                                              |  if ip = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  |                    IP-ID                     |  2 octets, if I = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  |                                              |
  /           Outer IP header fields             /  variable,
  |                                              |  if ip2 = 1
   ..... ..... ..... ..... ..... ..... ..... .....
  |                                              |
  /         RTP header flags and fields          /  variable,
  |                                              |  if rtp = 1
   ..... ..... ..... ..... ..... ..... ..... .....
```

S, R-TS, I, ip, rtp, ip2: Indicate presence of fields as shown to
the right of each field above.

Tsc: Tsc = 0 indicates that TS is not scaled;
     Tsc = 1 indicates that TS is scaled according to section
     4.5.3, using value(TS_STRIDE).
     Context(Tsc) is always 1.  If scaling is not desired, the
     compressor will establish TS_STRIDE = 1.

SN: See the beginning of section 5.7.

TS: Variable number of bits of TS, encoded according to
    section 4.5.6.  See the beginning of section 5.7.

IP-ID: See the beginning of section 5.7.

   Inner IP header flags

      These correspond to the inner IP header if there are two, and the
      single IP header otherwise.

         0     1     2     3     4     5     6     7
       ..... ..... ..... ..... ..... ..... ..... .....
      | TOS | TTL | DF  | PR  | IPX | NBO | RND | ip2 |  if ip = 1
       ..... ..... ..... ..... ..... ..... ..... .....

         TOS, TTL, PR, IPX: Indicates presence of fields as shown to the
            right of the field in question below.

         DF: Don't Fragment bit of IP header.

         NBO: Indicates whether the octets of hdr(IP identifier) of this IP
         header are swapped before compression and after decompression.

         NBO = 1 indicates that the octets need not be swapped.  NBO = 0
         indicates that the octets are to be swapped.  See section 4.5.5.

         RND: Indicates whether hdr(IP identifier) is not to be compressed
         but instead sent as-is in compressed headers.

         IP2: Indicates presence of Outer IP header fields.  Unless the
         static context contains two IP headers, IP2 is always zero.

   Inner IP header fields

       ..... ..... ..... ..... ..... ..... ..... .....
      |          Type of Service/Traffic Class          |  if TOS = 1
       ..... ..... ..... ..... ..... ..... ..... .....
      |           Time to Live/Hop Limit                |  if TTL = 1
       ..... ..... ..... ..... ..... ..... ..... .....
      |           Protocol/Next Header                  |  if PR = 1
       ..... ..... ..... ..... ..... ..... ..... .....
      /           IP extension headers                  /  variable,
       ..... ..... ..... ..... ..... ..... ..... .....     if IPX = 1

         Type of Service/Traffic Class: That field in the uncompressed IP
         header (absolute value).

         Time to Live/Hop Limit: That field in the uncompressed IP header.

         Protocol/Next Header: That field in the uncompressed IP header.

         IP extension header(s): According to section 5.8.5.

   Outer IP header flags

      The fields in this part of the Extension 3 header refer to the
      outermost IP header:

          0     1     2     3     4     5     6     7
        ..... ..... ..... ..... ..... ..... ..... .....  | TOS2| TTL2|
        DF2 | PR2 |IPX2 |NBO2 |RND2 |  I2 |  if ip2 = 1
        ..... ..... ..... ..... ..... ..... ..... .....

      These flags are the same as the Inner IP header flags, but refer
      to the outer IP header instead of the inner IP header.  The
      following flag, however, has no counterpart in the Inner IP header
      flags:

         I2: Indicates presence of the IP-ID field.

   Outer IP header fields

        ..... ..... ..... ..... ..... ..... ..... .....
        |       Type of Service/Traffic Class           |  if TOS2 = 1
        ..... ..... ..... ..... ..... ..... ..... .....
        |          Time to Live/Hop Limit               |  if TTL2 = 1
        ..... ..... ..... ..... ..... ..... ..... .....
        |           Protocol/Next Header                |  if PR2 = 1
        ..... ..... ..... ..... ..... ..... ..... .....
        /          IP extension header(s)               /  variable,
        ..... ..... ..... ..... ..... ..... ..... .....     if IPX2 = 1
        |                  IP-ID                         |  2 octets,
        ..... ..... ..... ..... ..... ..... ..... .....     if I2 = 1

      The fields in this part of Extension 3 are as for the Inner IP
      header fields, but they refer to the outer IP header instead of
      the inner IP header.  The following field, however, has no
      counterpart among the Inner IP header fields:

         IP-ID: The IP Identifier field of the outer IP header, unless
         the inner header is an IPv6 header, in which case I2 is always
         zero.

RTP header flags and fields

```
     0     1     2     3     4     5     6     7
 ..... ..... ..... ..... ..... ..... ..... .....
|   Mode    |R-PT |  M  | R-X |CSRC | TSS | TIS |  if rtp = 1
 ..... ..... ..... ..... ..... ..... ..... .....
| R-P |              RTP PT                     |  if R-PT = 1
 ..... ..... ..... ..... ..... ..... ..... .....
/          Compressed CSRC list                /  if CSRC = 1
 ..... ..... ..... ..... ..... ..... ..... .....
/                 TS_STRIDE                     /  1-4 oct if TSS = 1
 ..... ..... ..... ..... ..... ..... ..... ....
/          TIME_STRIDE (milliseconds)           /  1-4 oct if TIS = 1
 ..... ..... ..... ..... ..... ..... ..... .....
```

Mode: Compression mode. 0 = Reserved,
                        1 = Unidirectional,
                        2 = Bidirectional Optimistic,
                        3 = Bidirectional Reliable.

R-PT, CSRC, TSS, TIS: Indicate presence of fields as shown to the
    right of each field above.

R-P: RTP Padding bit, absolute value (presumed zero if absent).

R-X: RTP eXtension bit, absolute value.

M: See the beginning of section 5.7.

RTP PT: Absolute value of RTP Payload type field.

Compressed CSRC list: See section 5.8.1.

TS_STRIDE: Predicted increment/decrement of the RTP Timestamp
field when it changes.  Encoded as in section 4.5.6.

TIME_STRIDE: Predicted time interval in milliseconds between
changes in the RTP Timestamp.  Also an indication that the
compressor desires to perform timer-based compression of the RTP
Timestamp field: see section 4.5.4.  Encoded as in section 4.5.6.

5.7.5.1.  RND flags and packet types

The values of the RND and RND2 flags are changed by sending UOR-2
headers with Extension 3, or IR-DYN headers, where the flag(s) have
their new values.  The establishment procedure of the flags is the
normal one for the current mode, i.e., in U-mode and O-mode the
values are repeated several times to ensure that the decompressor

   receives at least one.  In R-mode, the flags are sent until an
   acknowledgment for a packet with the new RND flag values is received.

   The decompressor updates the values of its RND and RND2 flags
   whenever it receives an UOR-2 with Extension 3 carrying values for
   RND or RND2, and the UOR-2 CRC verifies successful decompression.

   When an IPv4 header for which the corresponding RND flag has not been
   established to be 1 is present in the static context, the packet
   types R-1 and UO-1 MUST NOT be used.

   When no IPv4 header is present in the static context, or the RND
   flags for all IPv4 headers in the context have been established to be
   1, the packet types R-1-ID, R-1-TS, UO-1-ID, and UO-1-TS MUST NOT be
   used.

   While in the transient state in which an RND flag is being
   established, the packet types R-1-ID, R-1-TS, UO-1-ID, and UO-1-TS
   MUST NOT be used.  This implies that the RND flag(s) of Extension 3
   may have to be inspected before the exact format of a base header
   carrying an Extension 3 can be determined, i.e., whether a T-bit is
   present or not.

5.7.5.2.  Flags/Fields in context

   Some flags and fields in Extension 3 need to be maintained in the
   context of the decompressor.  Their values are established using the
   mechanism appropriate to the compression mode, unless otherwise
   indicated in the table below and in referred sections.

   Flag/Field        Initial value   Comment
   -------------------------------------------------------------------
     Mode            Unidirectional  See section 5.6

     NBO             1               See section 4.5.5
     RND             0               See sections 4.5.5, 5.7.5.1

     NBO2            1               As NBO, but for outer header
     RND2            0               As RND, but for outer header

     TS_STRIDE       1               See section 4.5.3
     TIME_STRIDE     0               See section 4.5.4
     Tsc             1               Tsc is always 1 in context;
                                     can be 0 only when an Extension 3
                                     is present. See the discussion of the
                                     TS field in the beginning of section
                                     5.7.

5.7.6.  Feedback packets and formats

   When the round-trip time between compressor and decompressor is
   large, several packets can be in flight concurrently.  Therefore,
   several packets may be received by the decompressor after feedback
   has been sent and before the compressor has reacted to feedback.
   Moreover, decompression may fail due to residual errors in the
   compressed header.

   Therefore,

   a) in O-mode, the decompressor SHOULD limit the rate at which
      feedback on successful decompression is sent (if it is sent at
      all);
   b) when decompression fails, feedback SHOULD be sent only when
      decompression of several consecutive packets has failed, and when
      this occurs, the feedback rate SHOULD be limited;
   c) when packets are received which belong to a rejected packet
      stream, the feedback rate SHOULD be limited.

   A decompressor MAY limit the feedback rate by sending feedback only
   for one out of every k packets provoking the same (kind of) feedback.
   The appropriate value of k is implementation dependent; k might be
   chosen such that feedback is sent 1-3 times per link round-trip time.

   See section 5.2.2 for a discussion concerning ways to provide
   feedback information to the compressor.

5.7.6.1.  Feedback formats for ROHC RTP

   This section describes the format for feedback information in ROHC
   RTP.  See also 5.2.2.

   Several feedback formats carry a field labeled SN.  The SN field
   contains LSBs of an RTP Sequence Number.  The sequence number to use
   is the sequence number of the header which caused the feedback
   information to be sent.  If that sequence number cannot be
   determined, for example when decompression fails, the sequence number
   to use is that of the last successfully decompressed header.  If no
   sequence number is available, the feedback MUST carry a SN-NOT-VALID
   option.  Upon reception, the compressor matches valid SN LSBs with
   the most recent header sent with a SN with matching LSBs.  The
   decompressor must ensure that it sends enough SN LSBs in its feedback
   that this correlation does not become ambiguous; e.g., if an 8-bit SN
   LSB field could wrap around within a round-trip time, the FEEDBACK-1
   format cannot be used.

FEEDBACK-1

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|               SN              |
+---+---+---+---+---+---+---+---+
```

   A FEEDBACK-1 is an ACK.  In order to send a NACK or a STATIC-NACK,
   FEEDBACK-2 must be used.  FEEDBACK-1 does not contain any mode
   information; FEEDBACK-2 must be used when mode information is
   required.

FEEDBACK-2

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|Acktype| Mode  |      SN       |
+---+---+---+---+---+---+---+---+
|               SN              |
+---+---+---+---+---+---+---+---+
/         Feedback options      /
+---+---+---+---+---+---+---+---+
```

      Acktype:  0 = ACK
                1 = NACK
                2 = STATIC-NACK
                3 is reserved (MUST NOT be used for parseability)

      Mode:     0 is reserved
                1 = Unidirectional mode
                2 = Bidirectional Optimistic mode
                3 = Bidirectional Reliable mode


      Feedback options: A variable number of feedback options, see
         section 5.7.6.2.  Options may appear in any order.

5.7.6.2.  ROHC RTP Feedback options

   A ROHC RTP Feedback option has variable length and the following
   general format:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|   Opt Type    |   Opt Len     |
+---+---+---+---+---+---+---+---+
/          option data          /  Opt Len octets
+---+---+---+---+---+---+---+---+
```

Sections 5.7.6.3-9 describe the currently defined ROHC RTP feedback
options.

5.7.6.3.  The CRC option

The CRC option contains an 8-bit CRC computed over the entire
feedback payload, without the packet type and code octet, but
including any CID fields, using the polynomial of section 5.9.1.  If
the CID is given with an Add-CID octet, the Add-CID octet immediately
precedes the FEEDBACK-1 or FEEDBACK-2 format.  For purposes of
computing the CRC, the CRC fields of all CRC options are zero.

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|  Opt Type = 1 |  Opt Len = 1  |
+---+---+---+---+---+---+---+---+
|             CRC               |
+---+---+---+---+---+---+---+---+
```

When receiving feedback information with a CRC option, the compressor
MUST verify the information by computing the CRC and comparing the
result with the CRC carried in the CRC option.  If the two are not
identical, the feedback information MUST be ignored.

5.7.6.4.  The REJECT option

The REJECT option informs the compressor that the decompressor does
not have sufficient resources to handle the flow.

```
+---+---+---+---+---+---+---+---+
|  Opt Type = 2 |  Opt Len = 0  |
+---+---+---+---+---+---+---+---+
```

When receiving a REJECT option, the compressor stops compressing the
packet stream, and should refrain from attempting to increase the
number of compressed packet streams for some time.  Any FEEDBACK
packet carrying a REJECT option MUST also carry a CRC option.

5.7.6.5.  The SN-NOT-VALID option

The SN-NOT-VALID option indicates that the SN of the feedback is not
valid.  A compressor MUST NOT use the SN of the feedback to find the
corresponding sent header when this option is present.

```
+---+---+---+---+---+---+---+---+
|  Opt Type = 3 |  Opt Len = 0  |
+---+---+---+---+---+---+---+---+
```

5.7.6.6.  The SN option

   The SN option provides 8 additional bits of SN.

```
   +---+---+---+---+---+---+---+---+
   |  Opt Type = 4 |  Opt Len = 1  |
   +---+---+---+---+---+---+---+---+
   |               SN              |
   +---+---+---+---+---+---+---+---+
```

5.7.6.7.  The CLOCK option

   The CLOCK option informs the compressor of the clock resolution of
   the decompressor.  This is needed to allow the compressor to estimate
   the jitter introduced by the clock of the decompressor when doing
   timer-based compression of the RTP Timestamp.

```
   +---+---+---+---+---+---+---+---+
   |  Opt Type = 5 |  Opt Len = 1  |
   +---+---+---+---+---+---+---+---+
   |      clock resolution (ms)    |
   +---+---+---+---+---+---+---+---+
```

   The smallest clock resolution which can be indicated is 1
   millisecond.  The value zero has a special meaning: it indicates that
   the decompressor cannot do timer-based compression of the RTP
   Timestamp.  Any FEEDBACK packet carrying a CLOCK option SHOULD also
   carry a CRC option.

5.7.6.8.  The JITTER option

   The JITTER option allows the decompressor to report the maximum
   jitter it has observed lately, using the following formula which is
   very similar to the formula for Max_Jitter_BC in section 4.5.4.

   Let observation window i contain the decompressor's best
   approximation of the sliding window of the compressor (see section
   4.5.4) when header i is received.

   $Max\_Jitter_i =$

   $$max \left\{ |(T_i - T_j) - ((a_i - a_j) / TIME\_STRIDE)|, \text{ for all headers } j \text{ in observation window } i \right\}$$

   $Max\_Jitter =$

   $$max \left\{ Max\_Jitter_i, \text{ for a large number of recent headers } i \right\}$$

This information may be used by the compressor to refine the formula
for determining k when doing timer-based compression of the RTP
Timestamp.

```
+---+---+---+---+---+---+---+---+
|  Opt Type = 6 |  Opt Len = 1  |
+---+---+---+---+---+---+---+---+
|           Max_Jitter          |
+---+---+---+---+---+---+---+---+
```

The decompressor MAY ignore the oldest observed values of
Max_Jitter_i.  Thus, the reported Max_Jitter may decrease.
Robustness will be reduced if the compressor uses a jitter estimate
which is too small.  Therefore, a FEEDBACK packet carrying a JITTER
option SHOULD also carry a CRC option.  Moreover, the compressor MAY
ignore decreasing Max_Jitter values.

5.7.6.9.  The LOSS option

The LOSS option allows the decompressor to report the largest
observed number of packets lost in sequence.  This information MAY be
used by the compressor to adjust the size of the reference window
used in U- and O-mode.

```
+---+---+---+---+---+---+---+---+
|  Opt Type = 7 |  Opt Len = 1  |
+---+---+---+---+---+---+---+---+
| longest loss event (packets)  |
+---+---+---+---+---+---+---+---+
```

The decompressor MAY choose to ignore the oldest loss events.  Thus,
the value reported may decrease.  Since setting the reference window
too small can reduce robustness, a FEEDBACK packet carrying a LOSS
option SHOULD also carry a CRC option.  The compressor MAY choose to
ignore decreasing loss values.

5.7.6.10.  Unknown option types

If an option type unknown to the compressor is encountered, it must
continue parsing the rest of the FEEDBACK packet, which is possible
since the length of the option is explicit, but MUST otherwise ignore
the unknown option.

5.7.6.11.  RTP feedback example

Feedback for CID 8 indicating an ACK for SN 17 and Bidirectional
Reliable mode can have the following formats.

Assuming small CIDs:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   1   1   1   0 | 0   1   1 |  feedback packet type, Code = 3
+---+---+---+---+---+---+---+---+
| 1   1   1   0 | 1   0   0   0 |  Add-CID octet with CID = 8
+---+---+---+---+---+---+---+---+
| 0   0 | 1   1 |  SN MSB = 0   |  AckType = ACK, Mode = Reliable
+---+---+---+---+---+---+---+---+
|           SN LSB = 17         |
+---+---+---+---+---+---+---+---+
```

The second, third, and fourth octet are handed to the compressor.

The FEEDBACK-1 format may also be used.  Assuming large CIDs:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   1   1   1   0 | 0   1   0 |  feedback packet type, Code = 2
+---+---+---+---+---+---+---+---+
| 0   0   0   0   1   0   0   0 |  large CID with value 8
+---+---+---+---+---+---+---+---+
|           SN LSB = 17         |
+---+---+---+---+---+---+---+---+
```

The second and third octet are handed to the compressor.

Assuming small CIDs:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 1   1   1   1   0 | 0   1   0 |  feedback packet type, Code = 2
+---+---+---+---+---+---+---+---+
| 1   1   1   0 | 1   0   0   0 |  Add-CID octet with CID = 8
+---+---+---+---+---+---+---+---+
|           SN LSB = 17         |
+---+---+---+---+---+---+---+---+
```

The second and third octet are handed to the compressor.

   Assuming small CIDs and CID 0 instead of CID 8:

     0   1   2   3   4   5   6   7
   +---+---+---+---+---+---+---+---+
   | 1   1   1   1   0 | 0   0   1 |  feedback packet type, Code = 1
   +---+---+---+---+---+---+---+---+
   |            SN LSB = 17        |
   +---+---+---+---+---+---+---+---+

      The second octet is handed to the compressor.

5.7.7.  RTP IR and IR-DYN packets

   The subheaders which are compressible are split into a STATIC part
   and a DYNAMIC part.  These parts are defined in sections 5.7.7.3
   through 5.7.7.7.

   The structure of a chain of subheaders is determined by each header
   having a Next Header, or Protocol, field.  This field identifies the
   type of the following header.  Each Static part below that is
   followed by another Static part contains the Next Header/Protocol
   field and allows parsing of the Static chain; the Dynamic chain, if
   present, is structured analogously.

   IR and IR-DYN packets will cause a packet to be delivered to upper
   layers if and only if the payload is non-empty.  This means that an
   IP/UDP/RTP packet where the UDP length indicates a UDP payload of
   size 12 octets cannot be represented by an IR or IR-DYN packet.  Such
   packets can instead be represented using the UNCOMPRESSED profile
   (section 5.10).

5.7.7.1.  Basic structure of the IR packet

   This packet type communicates the static part of the context, i.e.,
   the values of the constant SN functions.  It can optionally also
   communicate the dynamic part of the context, i.e., the parameters of
   nonconstant SN functions.  It can also optionally communicate the
   payload of an original packet, if any.

```
      0   1   2   3   4   5   6   7
     --- --- --- --- --- --- --- ---
    |          Add-CID octet        | if for small CIDs and CID != 0
    +---+---+---+---+---+---+---+---+
    | 1   1   1   1   1   1   0 | D |
    +---+---+---+---+---+---+---+---+
    |                               |
    /     0-2 octets of CID info    / 1-2 octets if for large CIDs
    |                               |
    +---+---+---+---+---+---+---+---+
    |            Profile            | 1 octet
    +---+---+---+---+---+---+---+---+
    |              CRC              | 1 octet
    +---+---+---+---+---+---+---+---+
    |                               |
    |          Static chain         | variable length
    |                               |
    +---+---+---+---+---+---+---+---+
    |                               |
    |          Dynamic chain        | present if D = 1, variable length
    |                               |
     - - - - - - - - - - - - - - - -
    |                               |
    |            Payload            | variable length
    |                               |
     - - - - - - - - - - - - - - - -
```
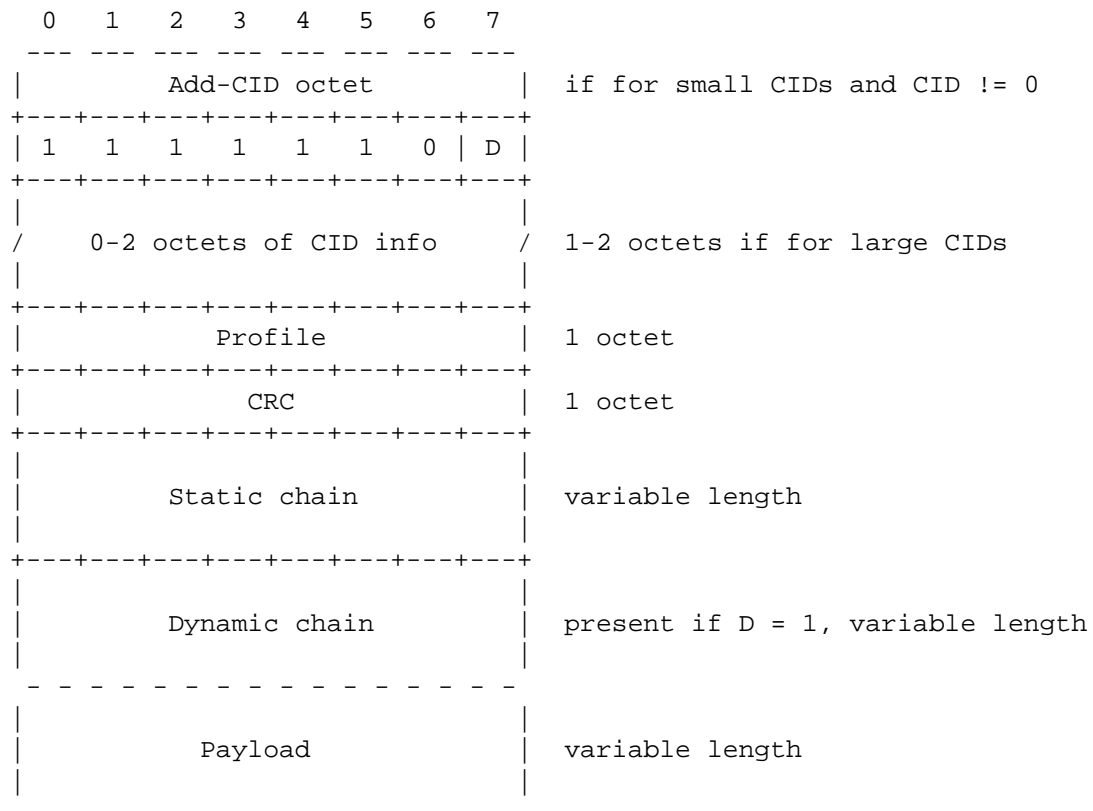
     D:   D = 1 indicates that the dynamic chain is present.

     Profile: Profile identifier, abbreviated as defined in section
        5.2.3.

     CRC: 8-bit CRC, computed according to section 5.9.1.

     Static chain: A chain of static subheader information.

     Dynamic chain: A chain of dynamic subheader information.  What
        dynamic information is present is inferred from the Static
        chain.

     Payload: The payload of the corresponding original packet, if any.
        The presence of a payload is inferred from the packet length.

5.7.7.2.  Basic structure of the IR-DYN packet

   This packet type communicates the dynamic part of the context, i.e.,
   the parameters of nonconstant SN functions.

```
     0   1   2   3   4   5   6   7
    --- --- --- --- --- --- --- ---
   :          Add-CID octet         : if for small CIDs and CID != 0
   +---+---+---+---+---+---+---+---+
   | 1   1   1   1   1   0   0   0 | IR-DYN packet type
   +---+---+---+---+---+---+---+---+
   :                               :
   /      0-2 octets of CID info   / 1-2 octets if for large CIDs
   :                               :
   +---+---+---+---+---+---+---+---+
   |             Profile           | 1 octet
   +---+---+---+---+---+---+---+---+
   |             CRC               | 1 octet
   +---+---+---+---+---+---+---+---+
   |                               |
   /          Dynamic chain        / variable length
   |                               |
   +---+---+---+---+---+---+---+---+
   :                               :
   /           Payload             / variable length
   :                               :
    - - - - - - - - - - - - - - - -
```
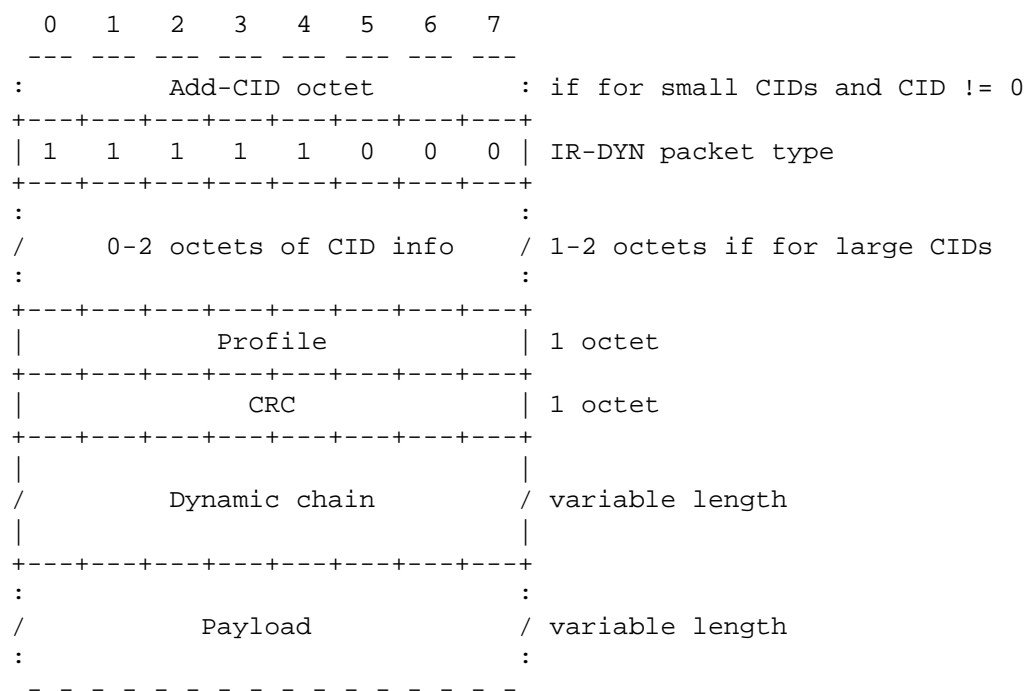
   Profile: Profile identifier, abbreviated as defined in section 5.2.3.

      CRC: 8-bit CRC, computed according to section 5.9.1.

         NOTE: As the CRC checks only the integrity of the header
         itself, an acknowledgment of this header does not signify that
         previous changes to the static chain in the context are also
         acknowledged.  In particular, care should be taken when IR
         packets that update an existing context are followed by IR-DYN
         packets.

   Dynamic chain: A chain of dynamic subheader information.  What
   dynamic information is present is inferred from the Static chain of
   the context.

   Payload: The payload of the corresponding original packet, if any.
   The presence of a payload is inferred from the packet length.

Note: The static and dynamic chains of IR or IR-DYN packets for
profile 0x0001 (ROHC RTP) MUST end with the static and dynamic parts
of an RTP header.  If not, the packet MUST be discarded and the
context MUST NOT be updated.

Note: The static or dynamic chains of IR or IR-DYN packets for
profile 0x0002 (ROHC UDP) MUST end with the static and dynamic parts
of a UDP header.  If not, the packet MUST be discarded and the
context MUST NOT be updated.

Note: The static or dynamic chains of IR or IR-DYN packets for
profile 0x0003 (ROHC ESP) MUST end with the static and dynamic parts
of an ESP header.  If not, the packet MUST be discarded and the
context MUST NOT be updated.

5.7.7.3.  Initialization of IPv6 Header [IPv6]

   Static part:

```
    +---+---+---+---+---+---+---+---+
    |  Version = 6  |Flow Label(msb)|   1 octet
    +---+---+---+---+---+---+---+---+
    /        Flow Label (lsb)       /   2 octets
    +---+---+---+---+---+---+---+---+
    |           Next Header         |   1 octet
    +---+---+---+---+---+---+---+---+
    /          Source Address       /   16 octets
    +---+---+---+---+---+---+---+---+
    /       Destination Address     /   16 octets
    +---+---+---+---+---+---+---+---+
```

   Dynamic part:

```
    +---+---+---+---+---+---+---+---+
    |          Traffic Class        |   1 octet
    +---+---+---+---+---+---+---+---+
    |           Hop Limit           |   1 octet
    +---+---+---+---+---+---+---+---+
    / Generic extension header list /   variable length
    +---+---+---+---+---+---+---+---+
```

   Eliminated:

      Payload Length

   Extras:

      Generic extension header list: Encoded according to section
      5.8.6.1, with all header items present in uncompressed form.

   CRC-DYNAMIC: Payload Length field (octets 5-6).

   CRC-STATIC: All other fields (octets 1-4, 7-40).

   CRC coverage for extension headers is defined in section 5.8.7.

   Note: The Next Header field indicates the type of the following
   header in the static chain, rather than being a copy of the Next
   Header field of the original IPv6 header.  See also section 5.7.7.8.

5.7.7.4.  Initialization of IPv4 Header [IPv4, section 3.1].

   Static part:

      Version, Protocol, Source Address, Destination Address.

```
      +---+---+---+---+---+---+---+---+
      |  Version = 4  |       0       |
      +---+---+---+---+---+---+---+---+
      |           Protocol            |
      +---+---+---+---+---+---+---+---+
      /         Source Address        /   4 octets
      +---+---+---+---+---+---+---+---+
      /       Destination Address     /   4 octets
      +---+---+---+---+---+---+---+---+
```

   Dynamic part:

      Type of Service, Time to Live, Identification, DF, RND, NBO,
      extension header list.

```
      +---+---+---+---+---+---+---+---+
      |         Type of Service       |
      +---+---+---+---+---+---+---+---+
      |          Time to Live         |
      +---+---+---+---+---+---+---+---+
      /          Identification       /   2 octets
      +---+---+---+---+---+---+---+---+
      | DF|RND|NBO|         0         |
      +---+---+---+---+---+---+---+---+
      / Generic extension header list /  variable length
      +---+---+---+---+---+---+---+---+
```

       Eliminated:

          IHL               (IP Header Length, must be 5)
          Total Length      (inferred in decompressed packets)
          MF flag           (More Fragments flag, must be 0)
          Fragment Offset   (must be 0)
          Header Checksum    (inferred in decompressed packets)
          Options, Padding  (must not be present)

          Extras:

             RND, NBO          See section 5.7.

             Generic extension header list: Encoded according to section
             5.8.6.1, with all header items present in uncompressed form.

       CRC-DYNAMIC: Total Length, Identification, Header Checksum
                    (octets 3-4, 5-6, 11-12).

       CRC-STATIC: All other fields (octets 1-2, 7-10, 13-20)

       CRC coverage for extension headers is defined in section 5.8.7.

       Note: The Protocol field indicates the type of the following header
       in the static chain, rather than being a copy of the Protocol field
       of the original IPv4 header.  See also section 5.7.7.8.

   5.7.7.5.  Initialization of UDP Header [RFC-768].

       Static part:

          +---+---+---+---+---+---+---+---+
          /           Source Port         /   2 octets
          +---+---+---+---+---+---+---+---+
          /         Destination Port      /   2 octets
          +---+---+---+---+---+---+---+---+

       Dynamic part:

          +---+---+---+---+---+---+---+---+
          /            Checksum           /   2 octets
          +---+---+---+---+---+---+---+---+

Eliminated:

Length

The Length field of the UDP header MUST match the Length field(s)
of the preceding subheaders, i.e., there must not be any padding
after the UDP payload that is covered by the IP Length.

CRC-DYNAMIC: Length field, Checksum (octets 5-8).

CRC-STATIC: All other fields (octets 1-4).

5.7.7.6.  Initialization of RTP Header [RTP].

Static part:

SSRC.

```
   0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+
 /             SSRC             /   4 octets
 +---+---+---+---+---+---+---+---+
```

Dynamic part:

P, X, CC, PT, M, sequence number, timestamp, timestamp stride,
CSRC identifiers.

```
   0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+
 | V=2   | P | RX|     CC        | (RX is NOT the RTP X bit)
 +---+---+---+---+---+---+---+---+
 | M |          PT              |
 +---+---+---+---+---+---+---+---+
 /      RTP Sequence Number     / 2 octets
 +---+---+---+---+---+---+---+---+
 /   RTP Timestamp (absolute)   / 4 octets
 +---+---+---+---+---+---+---+---+
 /       Generic CSRC list      / variable length
 +---+---+---+---+---+---+---+---+
 : Reserved  | X |  Mode  |TIS|TSS: if RX = 1
 +---+---+---+---+---+---+---+---+
 :            TS_Stride         : 1-4 octets, if TSS = 1
 +---+---+---+---+---+---+---+---+
 :           Time_Stride        : 1-4 octets, if TIS = 1
 +---+---+---+---+---+---+---+---+
```

   Eliminated:

      Nothing.

   Extras:

      RX: Controls presence of extension.

      Mode: Compression mode. 0 = Reserved,
                              1 = Unidirectional,
                              2 = Bidirectional Optimistic,
                              3 = Bidirectional Reliable.

   X: Copy of X bit from RTP header (presumed 0 if RX = 0)

   Reserved: Set to zero when sending, ignored when received.

   Generic CSRC list: CSRC list encoded according to section
         5.8.6.1, with all CSRC items present.

   CRC-DYNAMIC: Octets containing M-bit, sequence number field,
                and timestamp (octets 2-8).

   CRC-STATIC: All other fields (octets 1, 9-12, original CSRC list).

5.7.7.7.  Initialization of ESP Header [ESP, section 2]

   This is for the case when the NULL encryption algorithm [NULL] is NOT
   being used with ESP, so that subheaders after the ESP header are
   encrypted (see 5.12).  See 5.8.4.3 for compression of the ESP header
   when NULL encryption is being used.

   Static part:

      +---+---+---+---+---+---+---+---+
      /               SPI             /   4 octets
      +---+---+---+---+---+---+---+---+

   Dynamic part:

      +---+---+---+---+---+---+---+---+
      /        Sequence Number        /   4 octets
      +---+---+---+---+---+---+---+---+

   Eliminated:

      Other fields are encrypted, and can neither be located nor
      compressed.

CRC-DYNAMIC: Sequence number (octets 5-8)

CRC-STATIC: All other octets.

Note: No encrypted data is considered to be part of the header for
purposes of computing the CRC, i.e., octets after the eight octet are
not considered part of the header.

5.7.7.8.  Initialization of Other Headers

Headers not explicitly listed in previous subsections can be
compressed only by making them part of an extension header chain
following an IPv4 or IPv6 header, see section 5.8.

5.8.  List compression

Header information from the packet stream to be compressed can be
structured as an ordered list, which is largely constant between
packets.  The generic structure of such a list is as follows.

```
        +--------+--------+--- ...--+--------+
  list: | item 1 | item 2 |         | item n |
        +--------+--------+--- ...--+--------+
```

This section describes the compression scheme for such information.
The basic principles of list-based compression are the following:

1) While the list is constant, no information about the list is sent
   in compressed headers.

2) Small changes in the list are represented as additions (Insertion
   scheme), or deletions (Removal scheme), or both (Remove Then
   Insert scheme).

3) The list can also be sent in its entirety (Generic scheme).

There are two kinds of lists: CSRC lists in RTP packets, and
extension header chains in IP packets (both IPv4 and IPv6).

IPv6 base headers and IPv4 headers cannot be part of an extension
header chain.  Headers which can be part of extension header chains
include

a) the AH header
b) the null ESP header
c) the minimal encapsulation header [RFC2004, section 3.1]
d) the GRE header [GRE1, GRE2]
e) IPv6 extension headers.

The table-based item compression scheme (5.8.1), which reduces the
size of each item, is described first.  Then it is defined which
reference list to use in the insertion and removal schemes (5.8.2).
List encoding schemes are described in section 5.8.3, and a few
special cases in section 5.8.4.  Finally, exact formats are described
in sections 5.8.5-5.8.6.

5.8.1.  Table-based item compression

   The Table-based item compression scheme is a way to compress
   individual items sent in compressed lists.  The compressor assigns
   each item in a list a unique identifier Index.  The compressor
   conceptually maintains a table with all items, indexed by Index.  The
   (Index, item) pair is sent together in compressed lists until the
   compressor gains enough confidence that the decompressor has observed
   the mapping between the item and its Index.  Such confidence is
   obtained by receiving an acknowledgment from the decompressor in R-
   mode, and in U/O-mode by sending L (Index, item) pairs (not
   necessarily consecutively).  After that, the Index alone is sent in
   compressed lists to indicate the corresponding item.  The compressor
   may reassign an existing Index to a new item, and then needs to re-
   establish the mapping in the same manner as above.

   The decompressor conceptually maintains a table that contains all
   (Index, item) pairs it knows about.  The table is updated whenever an
   (Index, item) pair is received (and decompression is verified by a
   CRC).  The decompressor retrieves the item from the table whenever an
   Index without an accompanying item is received.

5.8.1.1.  Translation table in R-mode

   At the compressor side, an entry in the Translation Table has the
   following structure.

                    +-------+------+---------------+
           Index i | Known | item | SN1, SN2, ... |
                    +-------+------+---------------+

   The Known flag indicates whether the mapping between Index i and item
   has been established, i.e., if Index i alone can be sent in
   compressed lists.  Known is initially zero.  It is also set to zero
   whenever Index i is assigned to a new item.  Known is set to one when
   the corresponding (Index, item) pair is acknowledged.
   Acknowledgments are based on the RTP Sequence Number, so a list of
   RTP Sequence Numbers of all packets which contain the (Index, item)
   pair is included in the translation table.  When a packet with a
   sequence number in the sequence number list is acknowledged, the
   Known flag is set, and the sequence number list can be discarded.

   Each entry in the Translation Table at the decompressor side has the
   following structure:

```
             +-------+------+
   Index i | Known | item |
             +-------+------+
```

   All Known fields are initialized to zero.  Whenever the decompressor
   receives an (Index, item) pair, it inserts item into the table at
   position Index and sets the Known flag in that entry to one.  If an
   index without an accompanying item is received for which the Known
   flag is zero, the header MUST be discarded and a NACK SHOULD be sent.

5.8.1.2.  Translation table in U/O-modes

   At the compressor side, each entry in the Translation Table has the
   following structure:

```
             +-------+------+---------+
   Index | Known | item | Counter |
             +-------+------+---------+
```

   The Index, Known, and item fields have the same meaning as in section
   5.8.1.1.

   Known is set when the (Index, item) pair has been sent in L
   compressed lists (not necessarily consecutively).  The Counter field
   keeps track of how many times the pair has been sent.  Counter is set
   to 0 for each new entry added to the table, and whenever Index is
   assigned to a new item.  Counter is incremented by 1 whenever an
   (Index, item) pair is sent.  When the counter reaches L, the Known
   field is set and after that only the Index needs to be sent in
   compressed lists.

   At the decompressor side, the Translation Table is the same as the
   Translation Table defined in R-mode.

5.8.2.  Reference list determination

   In reference based compression schemes (i.e., addition or deletion
   based schemes), compression and decompression of a list (curr_list)
   are based on a reference list (ref_list) which is assumed to be
   present in the context of both compressor and decompressor.  The
   compressed list is an encoding of the differences between curr_list
   and ref_list.  Upon reception of a compressed list, the decompressor
   applies the differences to its reference list in order to obtain the
   original list.

   To identify the reference list (to be) used, each compressed list
   carries an identifier (ref_id).  The reference list is established by
   different methods in R-mode and U/O-mode.

5.8.2.1.  Reference list in R-mode and U/O-mode

   In R-mode, the choice of reference list is based on acknowledgments,
   i.e., the compressor uses as ref_list the latest list which has been
   acknowledged by the decompressor.  The ref_list is updated only upon
   receiving an acknowledgment.  The least significant bits of the RTP
   Sequence Number of the acknowledged packet are used as the ref_id.

   In U/O-mode, a sequence of identical lists are considered as
   belonging to the same generation and are all assigned the same
   generation identifier (gen_id).  Gen_id increases by 1 each time the
   list changes and is carried in compressed and uncompressed lists that
   are candidates for being used as reference lists.  Normally, Gen_id
   must have been repeated in at least L headers before the list can be
   used as a ref_list.  However, some acknowledgments may be sent in O-
   mode (and also in U-mode), and whenever an acknowledgment for a
   header is received, the list of that header is considered known and
   need not be repeated further.  The least significant bits of the
   Gen_id is used as the ref_id in U/O-mode.

   The logic of the compressor and decompressor for reference based list
   compression is similar to that for SN and TS.  The principal
   difference is that the decompressor maintains a sliding window with
   candidates for ref_list, and retrieves ref_list from the sliding
   window using the ref_id of the compressed list.

   Logic of compressor:

   a) In the IR state, the compressor sends Generic lists (see 5.8.5)
      containing all items of the current list in order to establish or
      refresh the context of the decompressor.

      In R-mode, such Generic lists are sent until a header is
      acknowledged.  The list of that header can be used as a reference
      list to compress subsequent lists.

      In U/O-mode, the compressor sends generation identifiers with the
      Generic lists until

      1) a generation identifier has been repeated L times, or

      2) an acknowledgment for a header carrying a generation identifier
         has been received.

The repeated (1) or acknowledged (2) list can be used as a
reference list to compress subsequent lists and is kept together
with its generation identifier.

b) When not in the IR state, the compressor moves to the FO state
   when it observes a difference between curr_list and the previous
   list.  It sends compressed lists based on ref_list to update the
   context of the decompressor.  (However, see d).)

   In R-mode, the compressor keeps sending compressed lists using the
   same reference until it receives an acknowledgment for a packet
   containing the newest list.  The compressor may then move to the
   SO state with regard to the list.

   In U/O-mode, the compressor keeps sending compressed lists with
   generation identifiers until

   1) a generation identifier has been repeated L times, or

   2) an acknowledgment for a header carrying the latest generation
      identifier has been received.

   The repeated or acknowledged list is used as the future reference
   list.  The compressor may move to the SO state with regard to the
   list.

c) In R-mode, the compressor maintains a sliding window containing
   the lists which have been sent to update the context of the
   decompressor and have not yet been acknowledged.  The sliding
   window shrinks when an acknowledgment arrives: all lists sent
   before the acknowledged list are removed.  The compressor may use
   the Index to represent items of lists in the sliding window.

   In U/O-mode, the compressor needs to store

   1) the reference list and its generation identifier, and

   2) if the current generation identifier is different from the
      reference generation, the current list and the sequence
      numbers with which the current list has been sent.

   (2) is needed to determine if an acknowledgment concerns the
       latest generation.  It is not needed in U-mode.

d) In U/O-mode, the compressor may choose to not send a generation
   identifier with a compressed list.  Such lists without generation
   identifiers are not assigned a new generation identifier and must

not be used as future reference lists.  They do not update the
context.  This feature is useful when a new list is repeated few
times and the list then reverts back to its old value.

Logic of decompressor:

e) In R-mode, the decompressor acknowledges all received uncompressed
   or compressed lists which establish or update the context.  (Such
   compressed headers contain a CRC.)

   In O-mode, the decompressor MAY acknowledge a list with a new
   generation identifier, see section 5.4.2.2.

   In U-mode, the decompressor MAY acknowledge a list sent in an IR
   packet, see section 5.3.2.3.

f) The decompressor maintains a sliding window which contains the
   lists that may be used as reference lists.

   In R-mode, the sliding window contains lists which have been
   acknowledged but not yet used as reference lists.

   In U/O-mode, the sliding window contains at most one list per
   generation.  It contains all generations seen by the decompressor
   newer than the last generation used as a reference.

g) When the decompressor receives a compressed list, it retrieves the
   proper ref_list from the sliding window based on the ref_id, and
   decompresses the compressed list obtaining curr_list.

   In R-mode, curr_list is inserted into the sliding window if an
   acknowledgment is sent for it.  The sliding window is shrunk by
   removing all lists received before ref_list.

   In U/O-mode, curr_list is inserted into the sliding window
   together with its generation identifier if the compressed list had
   a generation identifier and the sliding window does not contain a
   list with that generation identifier.  All lists with generations
   older than ref_id are removed from the sliding window.

5.8.3.  Encoding schemes for the compressed list

Four encoding schemes for the compressed list are described here.
The exact formats of the compressed CSRC list and compressed IP
extension header list using these encoding schemes are described in
sections 5.8.5-5.8.6.

Generic scheme

    In contrast to subsequent schemes, this scheme does not rely on a
    reference list having been established.  The entire list is sent,
    using table based compression for each individual item.  The
    generic scheme is always used when establishing the context of the
    decompressor and may also be used at other times, as the
    compressor sees fit.

Insertion Only scheme

    When the new list can be constructed from ref_list by adding
    items, a list of the added items is sent (using table based
    compression), along with the positions in ref_list where the new
    items will be inserted.  An insertion bit mask indicates the
    insertion positions in ref_list.

    Upon reception of a list compressed according to the Insertion
    Only scheme, curr_list is obtained by scanning the insertion bit
    mask from left to right.  When a '0' is observed, an item is
    copied from the ref_list.  When a '1' is observed, an item is
    copied from the list of added items.  If a '1' is observed when
    the list of added items has been exhausted, an error has occurred
    and decompression fails: The header MUST NOT be delivered to upper
    layers; it should be discarded, and MUST NOT be acknowledged nor
    used as a reference.

    To construct the insertion bit mask and the list of added items,
    the compressor MAY use the following algorithm:

    1) An empty bit list and an empty Inserted Item list are generated
       as the starting point.

    2) Start by considering the first item of curr_list and ref_list.

    3) If curr_list has a different item than ref_list,

            a set bit (1) is appended to the bit list;
            the first item in curr_list (represented using table-based
            item compression) is appended to the Inserted Item list;
            advance to the next item of curr_list;

       otherwise,

            a zero bit (0) is appended to the bit list;

            advance to the next item of curr_list;
            advance to the next item of ref_list.

   4) Repeat 3) until curr_list has been exhausted.

   5) If the length of the bit list is less than the required bit
      mask length, append additional zeroes.

Removal Only scheme

   This scheme can be used when curr_list can be obtained by removing
   some items in ref_list.  The positions of the items which are in
   ref_list, but not in curr_list, are sent as a removal bit mask.

   Upon reception of the compressed list, the decompressor obtains
   curr_list by scanning the removal bit mask from left to right.
   When a '0' is observed, the next item of ref_list is copied into
   curr_list.  When a '1' is observed, the next item of ref_list is
   skipped over without being copied.  If a '0' is observed when
   ref_list has been exhausted, an error has occurred and
   decompression fails: The header MUST NOT be delivered to upper
   layers; it should be discarded, and MUST NOT be acknowledged nor
   used as a reference.

   To construct the removal bit mask and the list of added items, the
   compressor MAY use the following algorithm:

   1) An empty bit list is generated as the starting point.

   2) Start by considering the first item of curr_list and ref_list.

   3) If curr_list has a different item than ref_list,

      a set bit (1) is appended to the bit list;
      advance to the next item of ref_list;

   otherwise,

      a zero bit (0) is appended to the bit list;
      advance to the next item of curr_list;
      advance to the next item of ref_list.

   4) Repeat 3) until curr_list has been exhausted.

   5) If the length of the bit list is less than the required bit
      mask length, append additional ones.

Remove Then Insert scheme

   In this scheme, curr_list is obtained by first removing items from
   ref_list, and then inserting items into the resulting list.  A
   removal bit mask, an insertion bit mask, and a list of added items
   are sent.

   Upon reception of the compressed list, the decompressor processes
   the removal bit mask as in the Removal Only scheme.  The resulting
   list is then used as the reference list when the insertion bit
   mask and the list of added items are processed, as in the
   Insertion Only scheme.

5.8.4.  Special handling of IP extension headers

   In CSRC list compression, each CSRC is assigned an index.  In
   contrast, in IP extension header list compression an index is usually
   associated with a type of extension header.  When there is more than
   one IP header, there is more than one list of extension headers.  An
   index per type per list is then used.

   The association with a type means that a new index need not always be
   used each time a field in an IP extension header changes.  However,
   when a field in an extension header changes, the mapping between the
   index and the new value of the extension header needs to be
   established, except in the special handling cases defined in the
   following subsections.

5.8.4.1.  Next Header field

   The next header field in an IP header or extension header changes
   whenever the type of the immediately following header changes, e.g.,
   when a new extension header is inserted after it, when the immediate
   subsequent extension header is removed from the list, or when the
   order of extension headers is changed.  Thus it may not be uncommon
   that, for a given header, the next header field changes while the
   remaining fields do not change.

   Therefore, in the case that only the next header field changes, the
   extension header is considered to be unchanged and rules for special
   treatment of the change in the next header field are defined below.

   All communicated uncompressed extension header items indicate their
   own type in their Next Header field.  Note that the rules below
   explain how to treat the Next Header fields while showing the
   conceptual reference list as an exact recreation of the original
   uncompressed extension header list.

a) When a subsequent extension header is removed from the list, the
   new value of the next header field is obtained from the reference
   extension header list.  For example, assume that the reference
   header list (ref_list) consists of headers A, B and C (ref_ext_hdr
   A, B, C), and the current extension header list (curr_list) only
   consists of extension headers A and C (curr_ext_hdr A, C).  The
   order and value of the next header fields of these extension
   headers are as follows.

```
ref_list:
+--------+-----+    +--------+-----+    +--------+-----+
| type B |     |    | type C |     |    | type D |     |
+--------+     |    +--------+     |    +--------+     |
|              |    |              |    |              |
+--------------+    +--------------+    +--------------+
ref_ext_hdr A        ref_ext_hdr B       ref_ext_hdr C


 curr_list:
+--------+-----+    +--------+-----+
| type C |     |    | type D |     |
+--------+     |    +--------+     |
|              |    |              |
+--------------+    +--------------+
 curr_ext_hdr A        curr_ext_hdr C
```

   Comparing the curr_ext_hdr A in curr_list and the ref_ext_hdr A in
   ref_list, the value of next header field is changed from "type B"
   to "type C" because of the removal of extension header B.  The new
   value of the next header field in curr_ext_hdr A, i.e., "type C",
   does not need to be sent to the decompressor.  Instead, it is
   retrieved from the next header field of the removed ref_ext_hdr B.

b) When a new extension header is inserted after an existing
   extension header, the next header field in the communicated item
   will carry the type of itself, rather than the type of the header
   that follows.  For example, assume that the reference header list
   (ref_list) consists of headers A and C (ref_ext_hdr A, C), and the
   current header list (curr_list) consists of headers A, B and C
   (curr_ext_hdr A, B, C).  The order and the value of the next
   header fields of these extension headers are as follows.

ref_list:

```
+--------+-----+    +--------+-----+
| type C |     |    | type D |     |
+--------+     |    +--------+     |
|        |     |    |        |     |
+-------------+    +-------------+
 ref_ext_hdr A      ref_ext_hdr C
```

curr_list:

```
+--------+-----+    +--------+-----+    +--------+-----+
| type B |     |    | type C |     |    | type D |     |
+--------+     |    +--------+     |    +--------+     |
|        |     |    |        |     |    |        |     |
+-------------+    +-------------+    +-------------+
 curr_ext_hdr A     curr_ext_hdr B     curr_ext_hdr C
```
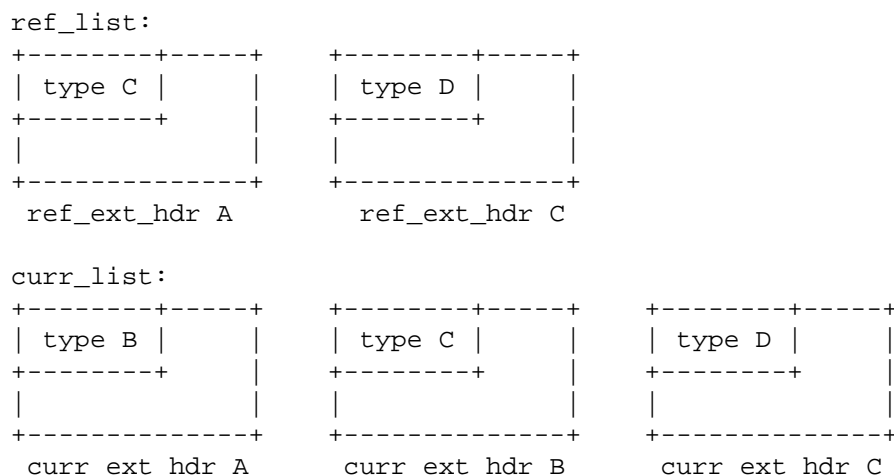
   Comparing the curr_list and the ref_list, the value of the next
   header field in extension header A is changed from "type C" to
   "type B".

   The uncompressed curr_ext_hdr B is carried in the compressed
   header list.  However, it carries "type B" instead of "type C" in
   its next header field.  When the decompressor inserts a new header
   after curr_ext_hdr A, the next header field of A is taken from the
   new header, and the next header field of the new header is taken
   from ref_ext_hdr A.

c) Some headers whose compression is defined in this document do not
   contain Next Header fields or do not have their Next Header field
   in the standard position (first octet of the header).  The GRE and
   ESP headers are such headers.  When sent as uncompressed items in
   lists, these headers are modified so that they do have a Next
   Header field as their first octet (see 5.8.4.3 and 5.8.4.4).  This
   is necessary to enable the decompressor to decode the item.

5.8.4.2.  Authentication Header (AH)

   The sequence number field in the AH [AH] contains a monotonically
   increasing counter value for a security association.  Therefore, when
   comparing curr_list with ref_list, if the sequence number in AH
   changes and SPI field does not change, the AH is not considered as
   changed.

   If the sequence number in the AH linearly increases as the RTP
   Sequence Number increases, and the compressor is confident that the
   decompressor has obtained the pattern, the sequence number in AH need
   not be sent.  The decompressor applies linear extrapolation to
   reconstruct the sequence number in the AH.

Otherwise, a compressed sequence number is included in the IPX
compression field in an Extension 3 of an UOR-2 header.

The authentication data field in AH changes from packet to packet and
is sent as-is.  If the uncompressed AH is sent, the authentication
data field is sent inside the uncompressed AH; otherwise, it is sent
after the compressed IP/UDP/RTP and IPv6 extension headers and before
the payload.  See beginning of section 5.7.

Note: The payload length field of the AH uses a different notion of
length than other IPv6 extension headers.

5.8.4.3.  Encapsulating Security Payload Header (ESP)

When the Encapsulating Security Payload Header (ESP) [ESP] is present
and an encryption algorithm other than NULL is being used, the UDP
and RTP headers are both encrypted and cannot be compressed.  The ESP
header thus ends the compressible header chain.  The ROHC ESP profile
defined in section 5.12 MAY be used for the stream in this case.

A special case is when the NULL encryption algorithm is used.  This
is the case when the ESP header is used for authentication only, and
not for encryption.  The payload is not encrypted by the NULL
encryption algorithm, so compression of the rest of the header chain
is possible.  The rest of this section describes compression of the
ESP header when the NULL encryption algorithm is used with ESP.

It is not possible to determine whether NULL encryption is used by
inspecting a header in the stream, this information is present only
at the encryption endpoints.  However, a compressor may attempt
compression under the assumption that the NULL encryption algorithm
is being used, and later abort compression when the assumption proves
to be false.

The compressor may, for example, inspect the Next Header fields and
the header fields supposed to be static in subsequent headers in
order to determine if NULL encryption is being used.  If these change
unpredictably, an encryption algorithm other than NULL is probably
being used and compression of subsequent headers SHOULD be aborted.
Compression of the stream is then either discontinued, or a profile
that compresses only up to the ESP header may be used (see 5.12).
While attempting to compress the header, the compressor should use
the SPI of the ESP header together with the destination IP address as
the defining fields for determining which packets belong to the
stream.

In the ESP header [ESP, section 2], the fields that can be compressed
are the SPI, the sequence number, the Next Header, and the padding
bytes if they are in the standard format defined in [ESP]. (As
always, the decompressor reinserts these fields based on the
information in the context.  Care must be taken to correctly reinsert
all the information as the Authentication Data must be verified over
the exact same information it was computed over.)

ESP header [ESP, section 2]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Security Parameters Index (SPI)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Payload Data (variable)                  |
~                                                               ~
|                                                               |
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               |         Padding (0-255 octets)                |
+-+-+-+-+-+-+-+-+                   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               | Pad Length   | Next Header    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Authentication Data                     |
+           (variable length, but assumed to be 12 octets)     +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
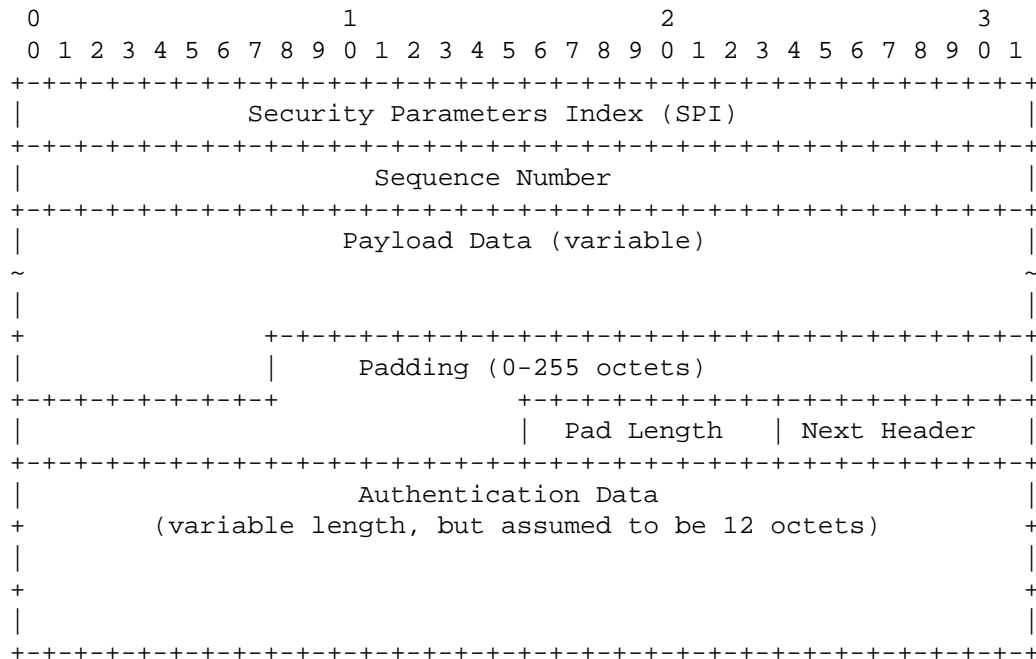
SPI: Static.  If it changes, it needs to be reestablished.

Sequence Number: Not sent when the offset from the sequence number
    of the compressed header is constant.  When the offset is not
    constant, the sequence number may be compressed by sending
    LSBs.  See 5.8.4.

Payload Data: This is where subsequent headers are to be found.
    Parsed according to the Next Header field.

Padding: The padding octets are assumed to be as defined in [ESP],
    i.e., to take the values 1, 2, ..., k, where k = Pad Length.
    If the padding in the static context has this pattern, padding
    in compressed headers is assumed to have this pattern as well
    and is removed.  If padding in the static context does not
    have this pattern, the padding is not removed.

        Pad Length: Dynamic.  Always sent.  14th octet from end of packet.

        Next Header: Static.  13th octet from end of packet.

     Authentication Data: Can have variable length, but when compression
     of NULL-encryption ESP header is attempted, it is assumed to have
     length 12 octets.

     The sequence number in ESP has the same behavior as the sequence
     number field in AH.  When it increases linearly, it can be compressed
     to zero bits.  When it does not increase linearly, a compressed
     sequence number is included in the IPX compression field in an
     Extension 3 of an UOR-2 header.

     The information which is part of an uncompressed item of a compressed
     list is the Next Header field, followed by the SPI and the Sequence
     Number.  Padding, Pad Length, Next Header, and Authentication Data
     are sent as-is at the end of the packet.  This means that the Next
     Header occurs in two places.

     Uncompressed ESP list item:

         +---+---+---+---+---+---+---+---+
         |          Next Header        ! 1 octet (see section 5.8.4.1)
         +---+---+---+---+---+---+---+---+
         /            SPI              / 4 octets
         +---+---+---+---+---+---+---+---+
         /        Sequence Number      / 4 octets
         +---+---+---+---+---+---+---+---+

        When sending Uncompressed ESP list items, all ESP fields near the
        the end of the packet are left untouched (Padding, Pad Length,
        Next Header, Authentication Data).

     A compressed item consists of a compressed sequence number.  When an
     item is compressed, Padding (if it follows the 1, 2, ..., k pattern)
     and Next Header are removed near the end of the packet.
     Authentication Data and Pad Length remain as-is near the end of the
     packet.

5.8.4.4.  GRE Header [RFC 2784, RFC 2890]

     The GRE header is a set of flags, followed by a mandatory Protocol
     Type and optional parts as indicated by the flags.

   The sequence number field in the GRE header contains a counter value
   for a GRE tunnel.  Therefore, when comparing curr_list with ref_list,
   if the sequence number in GRE changes, the GRE is not considered as
   changed.

   If the sequence number in the GRE header linearly increases as the
   RTP Sequence Number increases and the compressor is confident that
   the decompressor has received the pattern, the sequence number in GRE
   need not be sent.  The decompressor applies linear extrapolation to
   reconstruct the sequence number in the GRE header.

   Otherwise, a compressed sequence number is included in the IPX
   compression field in an Extension 3 of an UOR-2 header.

   The checksum data field in GRE, if present, changes from packet to
   packet and is sent as-is.  If the uncompressed GRE header is sent,
   the checksum data field is sent inside the uncompressed GRE header;
   otherwise, if present, it is sent after the compressed IP/UDP/RTP and
   IPv6 extension headers and before the payload.  See beginning of
   section 5.7.

   In order to allow simple parsing of lists of items, an uncompressed
   GRE header sent as an item in a list is modified from the original
   GRE header in the following manner: 1) the 16-bit Protocol Type field
   that encodes the type of the subsequent header using Ether types (see
   Ether types section in [ASSIGNED]) is removed.  2) A one-octet Next
   Header field is inserted as the first octet of the header.  The value
   of the Next Header field corresponds to GRE (this value is 47
   according to the Assigned Internet Protocol Number section of
   [ASSIGNED]) when the uncompressed item is to be inserted in a list,
   and to the type of the subsequent header when the uncompressed item
   is in a Generic list.  Note that this implies that only GRE headers
   with Ether types that correspond to an IP protocol number can be
   compressed.

   Uncompressed GRE list item:

       +---+---+---+---+---+---+---+---+
       |         Next Header         ! | 1 octet (see section 5.8.4.1)
       +---+---+---+---+---+---+---+---+
       / C |   | K | S |   |    Ver    | 1 octet
       +---+---+---+---+---+---+---+---+
       /           Checksum            / 2 octets, if C=1
       +---+---+---+---+---+---+---+---+
       /              Key              / 4 octets, if K=1
       +---+---+---+---+---+---+---+---+
       /        Sequence Number        / 4 octets, if S=1
       +---+---+---+---+---+---+---+---+

The bits left blank in the second octet are set to zero when
sending and ignored when received.

The fields Reserved0 and Reserved1 of the GRE header [GRE2] must
be all zeroes; otherwise, the packet cannot be compressed by this
profile.

5.8.5.  Format of compressed lists in Extension 3

5.8.5.1.  Format of IP Extension Header(s) field

In Extension 3 (section 5.7.5), there is a field called IP extension
header(s).  This section describes the format of that field.

```
      0     1     2     3     4     5     6     7
   +-----+-----+-----+-----+-----+-----+-----+-----+
   | CL  | ASeq| ESeq| Gseq|          res          | 1 octet
   +-----+-----+-----+-----+-----+-----+-----+-----+
   :    compressed AH Seq Number,  1 or 4 octets   :  if ASeq = 1
    ----- ----- ----- ----- ----- ----- ----- -----
   :    compressed ESP Seq Number, 1 or 4 octets   :  if Eseq = 1
    ----- ----- ----- ----- ----- ----- ----- -----
   :    compressed GRE Seq Number, 1 or 4 octets   :  if Gseq = 1
    ----- ----- ----- ----- ----- ----- ----- -----
   :    compressed header list, variable length    :  if CL = 1
    ----- ----- ----- ----- ----- ----- ----- -----


   ASeq: indicates presence of compressed AH Seq Number
   ESeq: indicates presence of compressed ESP Seq Number
   GSeq: indicates presence of compressed GRE Seq Number
   CL:   indicates presence of compressed header list
   res:  reserved; set to zero when sending, ignored when received
```

When Aseq, Eseq, or Gseq is set, the corresponding header item (AH,
ESP, or GRE header) is compressed.  When not set, the corresponding
header item is sent uncompressed or is not present.

The format of compressed AH, ESP and GRE Sequence Numbers can each be
either of the following:

```
    0   1   2   3   4   5   6   7      0   1   2   3   4   5   6   7
  +---+---+---+---+---+---+---+---+  +---+---+---+---+---+---+---+---+
  | 0 |   LSB of sequence number  |  | 1 |                         |
  +---+---+---+---+---+---+---+---+  +---+                         +
                                    |                             |
                                    +      LSB of sequence number +
                                    |                             |
                                    +                             +
                                    |                             |
                                    +---+---+---+---+---+---+---+---+
```
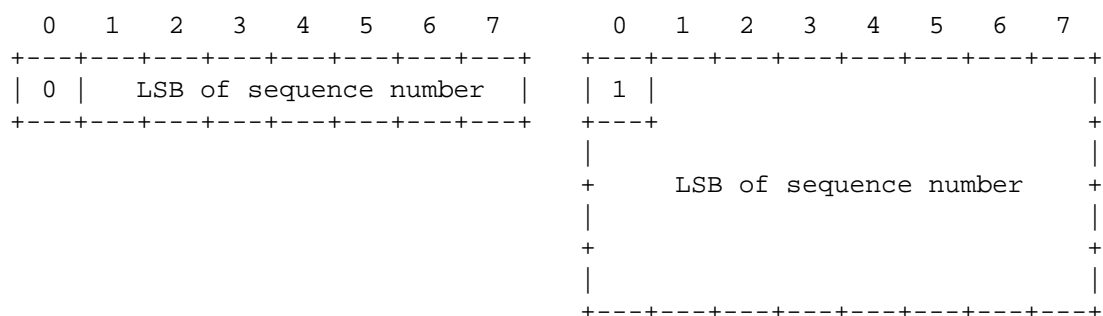
   The format of the compressed header list field is described in
   section 5.8.6.

5.8.5.2.  Format of Compressed CSRC List

   The Compressed CSRC List field in the RTP header part of an Extension
   3 (section 5.7.5) is as in section 5.8.6.

5.8.6.  Compressed list formats

   This section describes the format of compressed lists.  The format is
   the same for CSRC lists and header lists.  In CSRC lists, the items
   are CSRC identifiers; in header lists, they are uncompressed or
   compressed headers, as described in 5.8.4.2-4.

5.8.6.1.  Encoding Type 0 (generic scheme)

```
    0   1   2   3   4   5   6   7
  +---+---+---+---+---+---+---+---+
  | ET=0  |GP |PS |   CC = m      |
  +---+---+---+---+---+---+---+---+
  :           gen_id           :  1 octet, if GP = 1
  +---+---+---+---+---+---+---+---+
  |        XI 1, ..., XI m        |  m octets, or m * 4 bits
  /              --- --- --- ---/
  |            :    Padding    :  if PS = 0 and m is odd
  +---+---+---+---+---+---+---+---+
  |                             |
  /       item 1, ..., item n    /  variable
  |                             |
  +---+---+---+---+---+---+---+---+
```
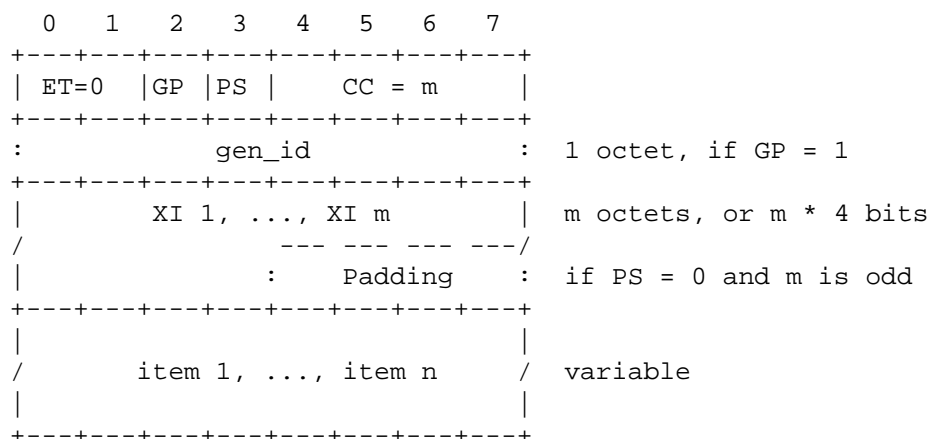
      ET: Encoding type is zero.

      PS: Indicates size of XI fields:
          PS = 0 indicates 4-bit XI fields;
          PS = 1 indicates 8-bit XI fields.

GP: Indicates presence of gen_id field.

CC: CSRC counter from original RTP header.

gen_id: Identifier for a sequence of identical lists.  It is
   present in U/O-mode when the compressor decides that it may use
   this list as a future reference list.

XI 1, ..., XI m: m XI items.  The format of an XI item is as
    follows:

```
             +---+---+---+---+
   PS = 0:   | X |   Index   |
             +---+---+---+---+


             0   1   2   3   4   5   6   7
             +---+---+---+---+---+---+---+---+
   PS = 1:   | X |           Index           |
             +---+---+---+---+---+---+---+---+
```

   X = 1 indicates that the item corresponding to the Index
        is sent in the item 0, ..., item n list.
   X = 0 indicates that the item corresponding to the Index is
        not sent.

When 4-bit XI items are used and m > 1, the XI items are placed in
octets in the following manner:

```
          0   1   2   3   4   5   6   7
        +---+---+---+---+---+---+---+---+
        |    XI k       |   XI k + 1    |
        +---+---+---+---+---+---+---+---+
```

Padding: A 4-bit padding field is present when PS = 0 and m is
odd.  The Padding field is set to zero when sending and ignored
when receiving.

Item 1, ..., item n:

   Each item corresponds to an XI with X = 1 in XI 1, ..., XI m.

5.8.6.2.  Encoding Type 1 (insertion only scheme)

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| ET=1  |GP |PS |      XI 1     |
+---+---+---+---+---+---+---+---+
:           gen_id              :  1 octet, if GP = 1
+---+---+---+---+---+---+---+---+
|              ref_id           |
+---+---+---+---+---+---+---+---+
/        insertion bit mask     /  1-2 octets
+---+---+---+---+---+---+---+---+
|              XI list          |  k octets, or (k - 1) * 4 bits
/              --- --- --- ---/
|              :    Padding     :  if PS = 0 and k is even
+---+---+---+---+---+---+---+---+
|                               |
/        item 1, ..., item n    /  variable
|                               |
+---+---+---+---+---+---+---+---+
```
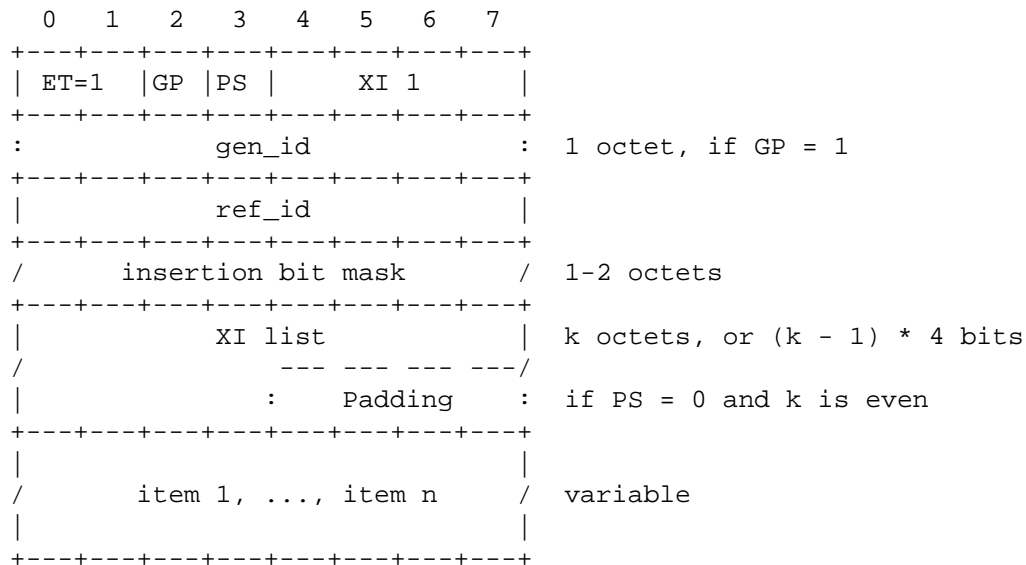
   Unless explicitly stated otherwise, fields have the same meaning and
   values as for encoding type 0.

      ET: Encoding type is one (1).

      XI 1: When PS = 0, the first 4-bit XI item is placed here.
            When PS = 1, the field is set to zero when sending, and
            ignored when receiving.

      ref_id: The identifier of the reference CSRC list used when the
            list was compressed.  It is the 8 least significant bits of
            the RTP Sequence Number in R-mode and gen_id (see section
            5.8.2) in U/O-mode.

      insertion bit mask: Bit mask indicating the positions where new
               items are to be inserted.  See Insertion Only scheme in
               section 5.8.3.  The bit mask can have either of the
               following two formats:

```
       0   1   2   3   4   5   6   7
     +---+---+---+---+---+---+---+---+
     | 0 |         7-bit mask        |   bit 1 is the first bit
     +---+---+---+---+---+---+---+---+


     +---+---+---+---+---+---+---+---+
     | 1 |                           |   bit 1 is the first bit
     +---+     15-bit mask         +
     |                             |   bit 7 is the last bit
     +---+---+---+---+---+---+---+---+
```
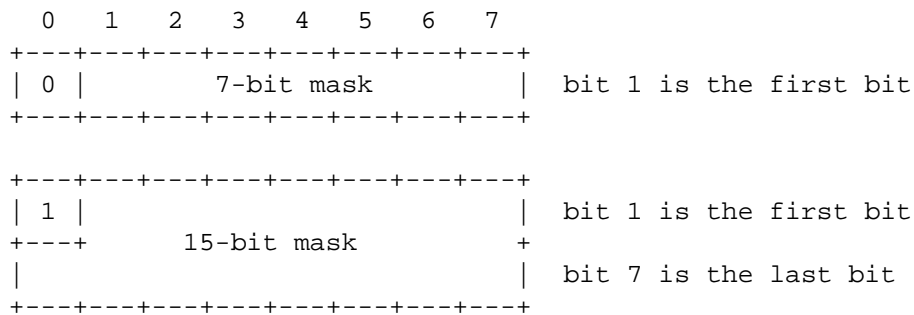
   XI list: XI fields for items to be inserted.  When the insertion
      bit mask has k ones, the total number of XI fields is k.  When
      PS = 1, all XI fields are in the XI list.  When PS = 0, the
      first XI field is in the XI 1 field, and the remaining k - 1
      XI fields are in the XI list.

   Padding: Present when PS = 0 and k is even.

   item 1, ..., item n: One item for each XI field with the X bit
      set.

5.8.6.3.  Encoding Type 2 (removal only scheme)

```
       0   1   2   3   4   5   6   7
     +---+---+---+---+---+---+---+---+
     | ET=2  |GP |res|      Count    |
     +---+---+---+---+---+---+---+---+
     :             gen_id            :  1 octet, if GP = 1
     +---+---+---+---+---+---+---+---+
     |             ref_id            |
     +---+---+---+---+---+---+---+---+
     /        removal bit mask       /  1-2 octets
     +---+---+---+---+---+---+---+---+
```
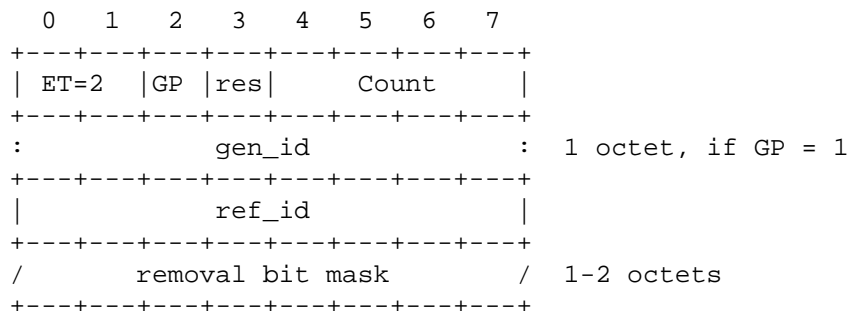
   Unless explicitly stated otherwise, fields have the same meaning
   and values as in section 5.8.5.2.

      ET: Encoding type is 2.

      res: Reserved.  Set to zero when sending, ignored when
         received.

      Count: Number of elements in ref_list.

removal bit mask: Indicates the elements in ref_list to be
    removed in order to obtain the current list.  See section
    5.8.3.  The removal bit mask has the same format as the
    insertion bit mask of section 5.8.6.3.

5.8.6.4.  Encoding Type 3 (remove then insert scheme)

See section 5.8.3 for a description of the Remove then insert
scheme.

```
   0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+
 | ET=3  |GP |PS |     XI 1      |
 +---+---+---+---+---+---+---+---+
 :             gen_id           :  1 octet, if GP = 1
 +---+---+---+---+---+---+---+---+
 |             ref_id           |
 +---+---+---+---+---+---+---+---+
 /        removal bit mask      /  1-2 octets
 +---+---+---+---+---+---+---+---+
 /       insertion bit mask     /  1-2 octets
 +---+---+---+---+---+---+---+---+
 |             XI list          |  k octets, or (k - 1) * 4 bits
 /              --- --- --- ---/
 |              :    Padding    :  if PS = 0 and k is even
 +---+---+---+---+---+---+---+---+
 |                              |
 /        item 1, ..., item n   /  variable
 |                              |
 +---+---+---+---+---+---+---+---+
```

The fields in this header have the same meaning and formats as in
section 5.8.5.2, except when explicitly stated otherwise below.

    ET: Encoding type is 3.

    removal bit mask: See section 5.8.6.3.

5.8.7.  CRC coverage for extension headers

All fields of extension headers are CRC-STATIC, with the following
exceptions which are CRC-DYNAMIC.

1) Entire AH header.
2) Entire ESP header.
3) Sequence number in GRE, Checksum in GRE

5.9.  Header compression CRCs, coverage and polynomials

   This chapter describes how to calculate the CRCs used in packet
   headers defined in this document.  (Note that another type of CRC is
   defined for reconstructed units in section 5.2.5.)

5.9.1.  IR and IR-DYN packet CRCs

   The CRC in the IR and IR-DYN packet is calculated over the entire IR
   or IR-DYN packet, excluding Payload and including CID or any Add-CID
   octet.  For purposes of computing the CRC, the CRC field in the
   header is set to zero.

   The initial content of the CRC register is to be preset to all 1's.

   The CRC polynomial to be used for the 8-bit CRC is:

      $C(x) = 1 + x + x^2 + x^8$

5.9.2.  CRCs in compressed headers

   The CRC in compressed headers is calculated over all octets of the
   entire original header, before compression, in the following manner.

   The octets of the header are classified as either CRC-STATIC or CRC-
   DYNAMIC, and the CRC is calculated over:

   1) the concatenated CRC-STATIC octets of the original header, placed
      in the same order as they appear in the original header, followed
      by

   2) the concatenated CRC-DYNAMIC octets of the original header, placed
      in the same order as they appear in the original header.

   The intention is that the state of the CRC computation after 1) will
   be saved.  As long as the CRC-STATIC octets do not change, the CRC
   calculation will then only need to process the CRC-DYNAMIC octets.

   In a typical RTP/UDP/IPv4 header, 25 octets are CRC-STATIC and 15 are
   CRC-DYNAMIC.  In a typical RTP/UDP/IPv6 header, 49 octets are CRC-
   STATIC and 11 are CRC-DYNAMIC.  This technique will thus reduce the
   computational complexity of the CRC calculation by roughly 60% for
   RTP/UDP/IPv4 and by roughly 80% for RTP/UDP/IPv6.

   Note: Whenever the CRC-STATIC fields change, the new saved CRC state
   after 1) is compared with the old state.  If the states are
   identical, the CRC cannot catch the error consisting in the
   decompressor not having updated the static context.  In U/O-mode the

   compressor SHOULD then for a while use packet types with another CRC
   length, for which there is a difference in CRC state, to ensure error
   detection.

   The initial content of the CRC register is preset to all 1's.

   The polynomial to be used for the 3 bit CRC is:

      $C(x) = 1 + x + x^3$

   The polynomial to be used for the 7 bit CRC is:

      $C(x) = 1 + x + x^2 + x^3 + x^6 + x^7$

   The CRC in compressed headers is calculated over the entire original
   header, before compression.

5.10.  ROHC UNCOMPRESSED -- no compression (Profile 0x0000)

   In ROHC, compression has not been defined for all kinds of IP
   headers.  Profile 0x0000 provides a way to send IP packets without
   compressing them.  This can be used for IP fragments, RTCP packets,
   and in general for any packet for which compression of the header has
   not been defined, is not possible due to resource constraints, or is
   not desirable for some other reason.

   After initialization, the only overhead for sending packets using
   Profile 0x0000 is the size of the CID.  When uncompressed packets are
   frequent, Profile 0x0000 should be associated with a CID with size
   zero or one octet.  There is no need to associate Profile 0x0000 with
   more than one CID.

5.10.1.  IR packet

   The initialization packet (IR packet) for Profile 0x0000 has the
   following format:

```
    0   1   2   3   4   5   6   7
  --- --- --- --- --- --- --- ---
 :          Add-CID octet        : if for small CIDs and (CID != 0)
 +---+---+---+---+---+---+---+---+
 | 1   1   1   1   1   1   0 |res|
 +---+---+---+---+---+---+---+---+
 :                               :
 /     0-2 octets of CID info    / 1-2 octets if for large CIDs
 :                               :
 +---+---+---+---+---+---+---+---+
 |           Profile = 0         | 1 octet
 +---+---+---+---+---+---+---+---+
 |              CRC              | 1 octet
 +---+---+---+---+---+---+---+---+
 :                               : (optional)
 /           IP packet           / variable length
 :                               :
  --- --- --- --- --- --- --- ---
```

     res: Always zero.

     Profile: 0.

     CRC: 8-bit CRC, computed using the polynomial of section 5.9.1.
     The CRC covers the first octet of the IR packet through the
     Profile octet of the IR packet, i.e., it does not cover the
     CRC itself or the IP packet.

     IP packet: An uncompressed IP packet may be included in the IR
     packet.  The decompressor determines if the IP packet is
     present by considering the length of the IR packet.

5.10.2.  Normal packet

   A Normal packet is a normal IP packet plus CID information.  When the
   channel uses small CIDs, and profile 0x0000 is associated with a CID
   > 0, an Add-CID octet is prepended to the IP packet.  When the
   channel uses large CIDs, the CID is placed so that it starts at the
   second octet of the Normal packet.

```
   0   1   2   3   4   5   6   7
 --- --- --- --- --- --- --- ---
:           Add-CID octet        : if for small CIDs and (CID != 0)
+---+---+---+---+---+---+---+---+
|    first octet of IP packet   |
+---+---+---+---+---+---+---+---+
:                               :
/    0-2 octets of CID info     / 1-2 octets if for large CIDs
:                               :
+---+---+---+---+---+---+---+---+
|                               |
/       rest of IP packet       / variable length
|                               |
+---+---+---+---+---+---+---+---+
```
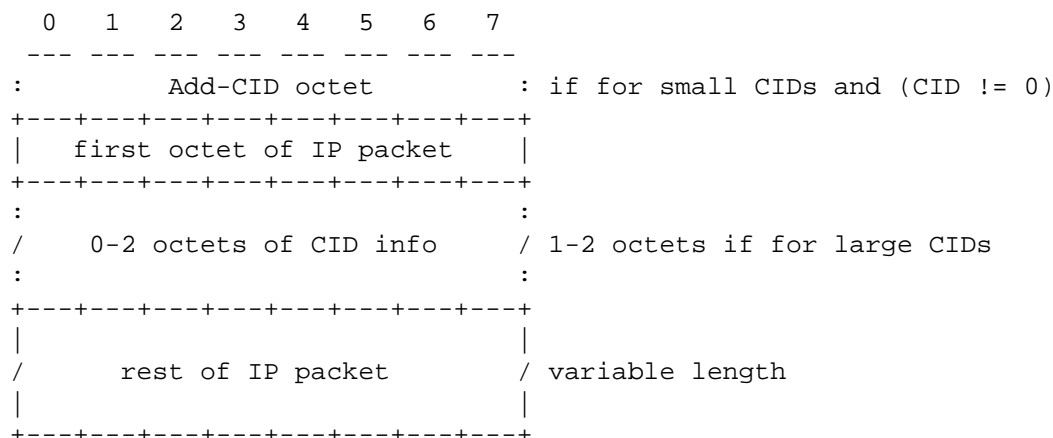
   Note that the first octet of the IP packet starts with the bit
   pattern 0100 (IPv4) or 0110 (IPv6).  This does not conflict with any
   reserved packet types.  Hence, no bits in addition to the CID are
   needed.  The profile is reasonably future-proof since problems do not
   occur until IP version 14.

5.10.3.  States and modes

   There are two modes in Profile 0x0000: Unidirectional mode and
   Bidirectional mode.  In Unidirectional mode, the compressor repeats
   the IR packet periodically.  In Bidirectional mode, the compressor
   never repeats the IR packet.  The compressor and decompressor always
   start in Unidirectional mode.  Whenever feedback is received, the
   compressor switches to Bidirectional mode.

   The compressor can be in either of two states: the IR state or the
   Normal state.  It starts in the IR state.

   a) IR state: Only IR packets can be sent.  After sending a small
      number of IR packets (only one when refreshing), the compressor
      switches to the Normal state.

   b) Normal state: Only Normal packets can be sent. When in
      Unidirectional mode, the compressor periodically transits back to
      the IR state.  The length of the period is implementation
      dependent, but should be fairly long.  Exponential backoff may be
      used.

   c) When feedback is received in any state, the compressor switches to
      Bidirectional mode.

The decompressor can be in either of two states: NO_CONTEXT or
FULL_CONTEXT.  It starts in NO_CONTEXT.

d) When an IR packet is received in the NO_CONTEXT state, the
   decompressor first verifies the packet using the CRC.  If the
   packet is OK, the decompressor 1) moves to the FULL_CONTEXT state,
   2) delivers the IP packet to upper layers if present, 3) MAY send
   an ACK.  If the packet is not OK, it is discarded without further
   action.

e) When any other packet is received in the NO_CONTEXT state, it is
   discarded without further action.

f) When an IR packet is received in the FULL_CONTEXT state, the
   packet is first verified using the CRC.  If OK, the decompressor
   1) delivers the IP packet to upper layers if present, 2) MAY send
   an ACK.  If the packet is not OK, no action is taken.

g) When a Normal packet is received in the FULL_CONTEXT state, the
   CID information is removed and the IP packet is delivered to upper
   layers.

5.10.4.  Feedback

The only kind of feedback in Profile 0x0000 is ACKs.  Profile 0x0000
MUST NOT be rejected.  Profile 0x0000 SHOULD be associated with at
most one CID.  ACKs use the FEEDBACK-1 format of section 5.2.  The
value of the profile-specific octet in the FEEDBACK-1 ACK is 0
(zero).

5.11.  ROHC UDP -- non-RTP UDP/IP compression (Profile 0x0002)

UDP/IP headers do not have a sequence number which is as well-behaved
as the RTP Sequence Number.  For UDP/IPv4, there is an IP-ID field
which may be echoed in feedback information, but when no IPv4 header
is present such feedback identification becomes problematic.

Therefore, in the ROHC UDP profile, the compressor generates a 16-bit
sequence number SN which increases by one for each packet received in
the packet stream.  This sequence number is thus relatively well-
behaved and can serve as the basis for most mechanisms described for
ROHC RTP.  It is called SN or UDP SN below.  Unless stated otherwise,
the mechanisms of ROHC RTP are used also for ROHC UDP, with the UDP
SN taking the role of the RTP Sequence Number.

The ROHC UDP profile always uses p = -1 when interpreting the SN,
since there will be no repetitions or reordering of the compressor-
generated SN.  The interpretation interval thus always starts with
(ref_SN + 1).

5.11.1.  Initialization

The static context for ROHC UDP streams can be initialized in either
of two ways:

1) By using an IR packet as in section 5.7.7.1, where the profile is
   two (2) and the static chain ends with the static part of an UDP
   packet.  At the compressor, UDP SN is initialized to a random
   value when the IR packet is sent.

2) By reusing an existing context where the existing static chain
   contains the static part of a UDP packet, e.g., the context of a
   stream compressed using ROHC RTP (profile 0x0001).  This is done
   with an IR-DYN packet (section 5.7.7.2) identifying profile
   0x0002, where the dynamic chain corresponds to the prefix of the
   existing static chain that ends with the UDP header.  UDP SN is
   initialized to the RTP Sequence Number if the earlier profile was
   profile 0x0001, and to a random number otherwise.

For ROHC UDP, the dynamic part of a UDP packet is different from
section 5.7.7.5: a two-octet field containing the UDP SN is added
after the Checksum field.  This affects the format of dynamic chains
in IR and IR-DYN packets.

Note: 2) can be used for packet streams which were initially assumed
to be RTP streams, so that compression started with profile 0x0001,
but were later found evidently not to be RTP streams.

5.11.2.  States and modes

ROHC UDP uses the same states and modes as ROHC RTP.  Mode
transitions and state logic are the same except when explicitly
stated otherwise.  Mechanisms dealing with fields in the RTP header
(except the RTP SN) are not used.  The decompressed UDP SN is never
included in any header delivered to upper layers.  The UDP SN is used
in place of the RTP SN in feedback.

5.11.3.  Packet types

   The general format of a ROHC UDP packet is the same as for ROHC RTP
   (see beginning of section 5.7).  Padding and CIDs are the same, as is
   the feedback packet type (5.7.6.1) and the feedback.  IR and IR-DYN
   packets (5.7.7) are changed as described in 5.11.1.

   The general format of compressed packets is also the same, but there
   are differences in specific formats and extensions as detailed below.
   The differences are caused by removal of all RTP specific information
   except the RTP SN, which is replaced by the UDP SN.

   Unless explicitly stated below, the packet formats are as in sections
   5.7.1-6.

   R-1

      The TS field is replaced by an IP-ID field.  The M flag has become
      part of IP-ID.  The X bit has moved.  The formats R-1-ID and R-1-
      TS are not used.

       0   1   2   3   4   5   6   7
     +---+---+---+---+---+---+---+---+
     | 1   0 |           SN          |
     +===+===+===+===+===+===+===+===+
     | X |            IP-ID          |
     +---+---+---+---+---+---+---+---+

   UO-1

      The TS field is replaced by an IP-ID field.  The M flag has become
      part of SN.  Formats UO-1-ID and UO-1-TS are not used.

       0   1   2   3   4   5   6   7
     +---+---+---+---+---+---+---+---+
     | 1   0 |          IP-ID        |
     +===+===+===+===+===+===+===+===+
     |        SN         |    CRC    |
     +---+---+---+---+---+---+---+---+

   UOR-2

New format:

```
   0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+
 | 1   1   0 |         SN        |
 +===+===+===+===+===+===+===+===+
 | X |           CRC             |
 +---+---+---+---+---+---+---+---+
```

5.11.4.  Extensions

   Extensions are as in 5.7.5, with the following exceptions:

   Extension 0:

```
     +---+---+---+---+---+---+---+---+
     | 0   0 |    SN     |   IP-ID   |
     +---+---+---+---+---+---+---+---+
```

   Extension 1:

```
     +---+---+---+---+---+---+---+---+
     | 0   1 |    SN     |   IP-ID   |
     +---+---+---+---+---+---+---+---+
     |               IP-ID           |
     +---+---+---+---+---+---+---+---+
```

   Extension 2:

```
     +---+---+---+---+---+---+---+---+
     | 1   0 |    SN     |   IP-ID2  |
     +---+---+---+---+---+---+---+---+
     |              IP-ID2           |
     +---+---+---+---+---+---+---+---+
     |               IP-ID          |
     +---+---+---+---+---+---+---+---+
```

        IP-ID2: For outer IP-ID field.

   Extension 3 is the same as Extension 3 in section 5.7.5, with the
   following exceptions.

   1) The initial flag octet has the following format:

```
        0     1     2     3     4     5     6     7
     +-----+-----+-----+-----+-----+-----+-----+-----+
     | 1     1 |  S  |    Mode   |  I  | ip  | ip2 |
     +-----+-----+-----+-----+-----+-----+-----+-----+
```

      Mode: Replaces R-TS and Tsc of 5.7.5.  Provides mode information
      as was earlier done in RTP header flags and fields.

      ip2: Replaces rtp bit of 5.7.5.  Moved here from the Inner IP
      header flags octet.

   2) The bit which was the ip2 flag in the Inner IP header flags in
      5.7.5 is reserved.  It is set to zero when sending and ignored
      when receiving.

5.11.5.  IP-ID

   Treated as in ROHC RTP, but the offset is from UDP SN.

5.11.6.  Feedback

   Feedback is as for ROHC RTP with the following exceptions:

   1) UDP SN replaces RTP SN in feedback.
   2) The CLOCK option (5.7.6.6) is not used.
   3) The JITTER option (5.7.6.7) is not used.

5.12.  ROHC ESP -- ESP/IP compression (Profile 0x0003)

   When the ESP header is being used with an encryption algorithm other
   than NULL, subheaders after the ESP header are encrypted and cannot
   be compressed.  Profile 0x0003 is for compression of the chain of
   headers up to and including the ESP header in this case.  When the
   NULL encryption algorithm is being used, other profiles can be used
   and could give higher compression rates.  See section 5.8.4.3.

   This profile is very similar to the ROHC UDP profile.  It uses the
   ESP sequence number as the basis for compression instead of a
   generated number, but is otherwise very similar to ROHC UDP.  The
   interpretation interval (value of p) for the ESP-based SN is as with
   ROHC RTP (profile 0x0001).  Apart from this, unless stated explicitly
   below, mechanisms and formats are as for ROHC UDP.

5.12.1.  Initialization

   The static context for ROHC ESP streams can be initialized in either
   of two ways:

   1) by using an IR packet as in section 5.7.7.1, where the profile is
      three (3) and the static chain ends with the static part of an ESP
      header.

   2) by reusing an existing context, where the existing static chain
      contains the static part of an ESP header.  This is done with an
      IR-DYN packet (section 5.7.7.2) identifying profile 0x0003, where
      the dynamic chain corresponds to the prefix of the existing static
      chain that ends with the ESP header.

   In contrast to ROHC UDP, no extra sequence number is added to the
   dynamic part of the ESP header: the ESP sequence number is the only
   element.

   Note: 2) can be used for streams where compression has been initiated
   under the assumption that NULL encryption was being used with ESP.
   When it becomes obvious that an encryption algorithm other than NULL
   is being used, the compressor may send an IR-DYN according to 2) to
   switch to profile 0x0003 without having to send an IR packet.

5.12.2.  Packet types

   The packet types for ROHC ESP are the same as for ROHC UDP, except
   that the ESP sequence number is used instead of the generated
   sequence number of ROHC UDP.  The ESP header is not part of any
   compressed list in ROHC ESP.

6.  Implementation issues

   This document specifies mechanisms for the protocol and leaves many
   details on the use of these mechanisms to the implementers.  This
   chapter is aimed to give guidelines, ideas and suggestions for
   implementing the scheme.

6.1.  Reverse decompression

   This section describes an OPTIONAL decompressor operation to reduce
   the number of packets discarded due to an invalid context.

   Once a context becomes invalid (e.g., when more consecutive packet
   losses than expected have occurred), subsequent compressed packets
   cannot immediately be decompressed correctly.  Reverse decompression
   aims at decompressing such packets later instead of discarding them,
   by storing them until the context has been updated and validated and
   then attempting decompression.

   Let the sequence of stored packets be i, i + 1, ..., i + k, where i
   is the first packet and i + k is the last packet before the context
   was updated.  The decompressor will attempt to recover the stored
   packets in reverse order, i.e., starting with i + k, and working back
   toward i.  When a stored packet has been reconstructed, its
   correctness is verified using its CRC.  Packets not carrying a CRC

must not be delivered to upper layers.  Packets where the CRC
succeeds are delivered to upper layers in their original order, i.e.,
i, i + 1, ..., i + k.

Note that this reverse decompression introduces buffering while
waiting for the context to be validated and thereby introduces
additional delay.  Thus, it should be used only when some amount of
delay is acceptable.  For example, for video packets belonging to the
same video frame, the delay in packet arrivals does not cause
presentation time delay.  Delay-insensitive streaming applications
can also be tolerant of such delay.  If the decompressor cannot
determine whether the application can tolerate delay, it should not
perform reverse decompression.

The following illustrates the decompression procedure in some detail:

1. The decompressor stores compressed packets that cannot be
   decompressed correctly due to an invalid context.

2. When the decompressor has received a context updating packet and
   the context has been validated, it proceeds to recover the last
   packet stored.  After decompression, the decompressor checks the
   correctness of the reconstructed header using the CRC.

3. If the CRC indicates successful decompression, the decompressor
   stores the complete packet and attempts to decompress the
   preceding packet.  In this way, the stored packets are recovered
   in reverse order until no compressed packets are left.  For each
   packet, the decompressor checks the correctness of the
   decompressed headers using the header compression CRC.

4. If the CRC indicates an incorrectly decompressed packet, the
   reverse decompression attempt MUST be terminated and all remaining
   uncompressed packets MUST be discarded.

5. Finally, the decompressor forwards all the correctly decompressed
   packets to upper layers in their original order.

6.2.  RTCP

RTCP is the RTP Control Protocol [RTP].  RTCP is based on periodic
transmission of control packets to all participants in a session,
using the same distribution mechanism as for data packets.  Its
primary function is to provide feedback from the data receivers on
the quality of the data distribution.  The feedback information may
be used for issues related to congestion control functions, and
directly useful for control of adaptive encodings.

In an RTP session there will be two types of packet streams: one with
the RTP header and application data, and one with the RTCP control
information.  The difference between the streams at the transport
level is in the UDP port numbers: the RTP port number is always even,
the RTCP port number is that number plus one and therefore always odd
[RTP, section 10].  The ROHC header compressor implementation has
several ways at hand to handle the RTCP stream:

1. One compressor/decompressor entity carrying both types of streams
   on the same channel, using CIDs to distinguish between them.  For
   sending a single RTP stream together with its RTCP packets on one
   channel, it is most efficient to set LARGE_CIDS to false, send the
   RTP packets with the implied CID 0 and use the Add-CID mechanism
   to send the RTCP packets.

2. Two compressor/decompressor entities, one for RTP and another one
   for RTCP, carrying the two types of streams on separate channels.
   This means that they will not share the same CID number space.

RTCP headers may simply be sent uncompressed using profile 0x0000.
More efficiently, ROHC UDP compression (profile 0x0002) can be used.

6.3.  Implementation parameters and signals

A ROHC implementation may have two kinds of parameters: configuration
parameters that are mandatory and must be negotiated between
compressor and decompressor peers, and implementation parameters that
are optional and, when used, stipulate how a ROHC implementation is
to operate.

Configuration parameters are mandatory and must be negotiated between
compressor and decompressor, so that they have the same values at
both compressor and decompressor, see section 5.1.1.

Implementation parameters make it possible for an external entity to
stipulate how an implementation of a ROHC compressor or decompressor
should operate.  Implementation parameters have local significance,
are optional to use and are thus not necessary to negotiate between
compressor and decompressor.  Note that this does not preclude
signaling or negotiating implementation parameters using lower layer
functionality in order to set the way a ROHC implementation should
operate.  Some implementation parameters are valid only at either of
compressor or decompressor.  Implementation parameters may further be
divided into parameters that allow an external entity to describe the
way the implementation should operate and parameters that allow an
external entity to trigger a specific event, i.e., signals.

6.3.1.  ROHC implementation parameters at compressor

   CONTEXT_REINITIALIZATION -- signal
   This parameter triggers a reinitialization of the entire context at
   the decompressor, both the static and the dynamic part.  The
   compressor MUST, when CONTEXT_REINITIALIZATION is triggered, back off
   to the IR state and fully reinitialize the context by sending IR
   packets with both the static and dynamic chains covering the entire
   uncompressed headers until it is reasonably confident that the
   decompressor contexts are reinitialized.  The context
   reinitialization MUST be done for all contexts at the compressor.
   This parameter may for instance be used to do context relocation at,
   e.g., a cellular handover that results in a change of compression
   point in the radio access network.

   NO_OF_PACKET_SIZES_ALLOWED -- value: positive integer
   This parameter may be set by an external entity to specify the number
   of packet sizes a ROHC implementation may use.  However, the
   parameter may be used only if PACKET_SIZES is not used by an external
   entity.  With this parameter set, the ROHC implementation at the
   compressor MUST NOT use more different packet sizes than the value
   this parameter stipulates.  The ROHC implementation must itself be
   able to determine which packet sizes will be used and describe these
   to an external entity using PACKET_SIZES_USED.  It should be noted
   that one packet size might be used for several header formats, and
   that the number of packet sizes can be reduced by employing padding
   and segmentation.

   NO_OF_PACKET_SIZES_USED _- value: positive integer
   This parameter is set by the ROHC implementation to indicate how many
   packet sizes it will actually use.  It can be set to a large value to
   indicate that no particular attempt is made to minimize that number.

   PACKET_SIZES_ALLOWED -- value: list of positive integers (bytes)
   This parameter, if set, governs which packet sizes in bytes may be
   used by the ROHC implementation.  Thus, packet sizes not in the set
   of values for this parameter MUST NOT be used.  Hence, an external
   entity can mandate a ROHC implementation to produce packet sizes that
   fit pre-configured lower layers better.  If this parameter is used to
   stipulate which packet sizes a ROHC implementation can use, the
   following rules apply:

   - A packet large enough to hold the entire IR header (both static and
     dynamic chain) MUST be part of the set of sizes, unless MRRU is set
     to a large enough value to allow segmentation.
   - The packet size likely to be used most frequently in the SO state
     SHOULD be part of the set.

   - The packet size likely to be used most frequently in the FO state
     SHOULD be part of the set.

   PACKET_SIZES_USED -- values: set of positive integers (bytes)
   This parameter describes which packet sizes a ROHC implementation
   uses if NO_OF_PACKET_SIZES_ALLOWED or PACKET_SIZES_ALLOWED is used by
   an external entity to stipulate how many packet sizes a ROHC
   implementation should use.  The information about used packet sizes
   (bytes) in this parameter, may then be used to configure lower
   layers.

   PAYLOAD_SIZES -_ values: set of positive integer values (bytes)
   This parameter is set by an external entity that wants to make use of
   the PACKET_SIZES_USED parameter to indicate which payload sizes can
   be expected.

   When a ROHC implementation has a limited set of allowed packet sizes,
   and the most preferable header format has a size that is not part of
   the set, it has the following options:

   - Choose the next larger header format from the allowed set.  This is
     probably the most efficient choice.
   - Use the most preferable header format as if there were no
     restrictions on size, and then add padding octets to complete a
     packet of the next larger size in the allowed set.
   - Use segmentation to fragment the packet into pieces that would make
     up packets of sizes that are permissible (possibly after the
     addition of padding to the last segment).

   It should be noted that even if the two last parameters introduce the
   possibility of restricting the number of packet sizes used, such
   restrictions will have a negative impact on compression performance.

6.3.2.  ROHC implementation parameters at decompressor

   MODE -- values: [U-mode, O-mode, R-mode]
   This parameter triggers a mode transition using the mechanism
   described in chapter 5 when the parameter changes value, i.e., to U-
   mode (Unidirectional mode), O-mode (Bidirectional Optimistic mode) or
   R-mode (Bidirectional Reliable mode).  The mode transition is made
   from the current mode to the new mode as signaled by the
   implementation parameter.  For example, if the current mode is
   Bidirectional Optimistic mode, MODE should have the value O-mode.  If
   the MODE is changed to R-mode, a mode transition MUST be made from
   Bidirectional Optimistic mode to Bidirectional Reliable mode.  MODE
   should not only serve as a trigger for mode transitions, but also
   make it visible which mode ROHC operates in.

CLOCK_RESOLUTION -- value: nonnegative integer
This parameter indicates the system clock resolution in units of
milliseconds.  A zero (0) value means that there is no clock
available.  If nonzero, this parameter allows the decompressor to use
timer-based TS compression (section 4.5.4) and SN wraparound
detection (section 5.3.2.2.4).  In this case, its specific value is
also significant for correctness of the algorithms.

REVERSE_DECOMPRESSION_DEPTH -- value: nonnegative integer
This parameter determines whether reverse decompression as described
in section 6.1 should be used or not, and if used, to what extent.
The value indicates the maximum number of packets that can be
buffered, and thus possibly be reverse decompressed by the
decompressor.  A zero (0) value means that reverse decompression MUST
NOT be used.

6.4.  Handling of resource limitations at the decompressor

   In a point-to-point link, the two nodes can agree on the number of
   compressed sessions they are prepared to support for this link.  It
   may, however, not be possible for the decompressor to accurately
   predict when it will run out of resources.  ROHC allows the
   negotiated number of contexts to be larger than could be accommodated
   in the worst case.  Then, as context resources are consumed, an
   attempt to set up a new context may be rejected by the decompressor,
   using the REJECT option of the feedback payload.

   Upon reception of a REJECT option, the compressor SHOULD wait for a
   while before attempting to compress additional streams destined for
   the rejecting node.

6.5.  Implementation structures

   This section provides some explanatory material on data structures
   that a ROHC implementation will have to maintain in one form or
   another.  It is not intended to constrain the implementations.

6.5.1.  Compressor context

   The compressor context consists of a static part and a dynamic part.
   The content of the static part is the same as the static chain
   defined in section 5.7.7.  The dynamic part consists of multiple
   elements which can be categorized into four types.

   a) Sliding Window (SW)
   b) Translation Table (TT)
   c) Flag
   d) Field

These elements may be common to all modes or mode specific.  The
following table summarizes all these elements.

```
+--------+--------------------------+------------+----------------+
|        |          Common to       | Specific to | Specific to    |
|        |          all modes       |  R-mode     |   U/O-mode     |
+--------+--------------------------+------------+----------------+
| SWs    | GSW                      | R_CSW      | UO_CSW         |
|        |                          | R_IESW     | UO_IESW        |
+--------+--------------------------+------------+----------------+
| TTs    |                          | R_CTT      | UO_CTT         |
|        |                          | R_IETT     | UO_IETT        |
+--------+--------------------------+------------+----------------+
| Flags  | UDP Chksum               |            | ACKED          |
|        | TSS, TIS                 |            |                |
|        | RND, RND2                |            |                |
|        | NBO, NBO2                |            |                |
+--------+--------------------------+------------+----------------+
| Fields | Profile                  |            | CSRC_REF_ID    |
|        | C_MODE                   |            | CSRC_GEN_ID    |
|        | C_STATE                  |            | CSRC_GEN_COUNT |
|        | C_TRANS                  |            | IPEH_REF_ID    |
|        | TS_STRIDE (if TSS = 1)   |            | IPEH_GEN_ID    |
|        | TS_OFFSET (if TSS = 1)   |            | IPEH_GEN_COUNT |
|        | TIME_STRIDE (if TIS = 1) |            |                |
|        | CURR_TIME (if TIS = 1)   |            |                |
|        | MAX_JITTER_CD (if TIS = 1)|            |                |
|        | LONGEST_LOSS_EVENT(O)    |            |                |
|        | CLOCK_RESOLUTION(O)      |            |                |
|        | MAX_JITTER(O)            |            |                |
+--------+--------------------------+------------+----------------+
```

1) GSW: Generic W_LSB Sliding Window

   Each element in GSW consists of all the dynamic fields in the
   dynamic chain (defined in section 5.7.7) plus the fields specified
   in a) but excluding the fields specified in b).

   a) Packet Arrival Time (if TIS = 1)
      Scaled RTP Time Stamp (if TSS = 1) (optional)
      Offset_i (if RND = 0) (optional)

   b) UDP Checksum, TS Stride, CSRC list, IPv6 Extension Headers

2) R_CSW: CSRC Sliding Window in R-mode

   R_IESW: IPv6 Extension Header Sliding Window in R-mode

      UO_CSW: CSRC Sliding Window in U/O-mode

      UO_IESW: IPv6 Extension Header Sliding Window in U/O-mode

      Each element in R_CSW, R_IESW, UO_CSW and UO_IESW is defined in
      section 6.5.3.

   3) R_CTT: CSRC Translation Table in R-mode

      R_IETT: IPv6 Extension Header Translation Table in U/O-mode

      UO_CTT: CSRC Translation Table in U/O-mode

      UO_IETT: IPv6 Extension Header Translation Table in U/O-mode

      Each element in R_CTT and R_IETT is defined in section 5.8.1.1.
      Each element in UO_CTT and UO_IETT is defined in section 5.8.1.2.

   4) ACKED: Indicates whether or not the decompressor has ever acked

   5) CURR_TIME: The current time value (used for context relocation
      when timer-based timestamp compression is used)

   6) All the other flags and fields are defined elsewhere in the ROHC
      document.

6.5.2.  Decompressor context

   The decompressor context consists of a static part and a dynamic
   part.  The content of the static part is the same as the static chain
   defined in section 5.7.7.  The dynamic part consists of multiple
   elements, one of which is the nonstatic reference header that
   includes all the nonstatic fields.  These nonstatic fields are the
   fields in the dynamic chain defined in section 5.7.7, excluding UDP
   Checksum and TS_Stride.  All the remaining elements can be
   categorized into four types:

   a) Sliding Window (SW)
   b) Translation Table (TT)
   d) Flag
   e) Field

   These elements may be mode specific or common to all modes.  The
   following table summarizes all these elements.

```
+--------+--------------------------+------------+----------------+
|        |         Common to        | Specific to | Specific to   |
|        |         all modes        |   R-mode   |   U/O-mode     |
+--------+--------------------------+------------+----------------+
| SWs    |                          | R_CSW      | UO_CSW         |
|        |                          | R_IESW     | UO_IESW        |
+--------+--------------------------+------------+----------------+
| TTs    |                          | R_CTT      | UO_CTT         |
|        |                          | R_IETT     | UO_IETT        |
+--------+--------------------------+------------+----------------+
| Flags  | UDP Checksum             |            | ACKED          |
|        | TSS, TIS                 |            |                |
|        | RND, RND2                |            |                |
|        | NBO, NBO2                |            |                |
+--------+--------------------------+------------+----------------+
| Fields | Profile                  |            | CSRC_GEN_ID    |
|        | D_MODE                   |            | IPEH_GEN_ID    |
|        | D_STATE                  |            | PRE_SN_V_REF   |
|        | D_TRANS                  |            |                |
|        | TS_STRIDE (if TSS = 1)   |            |                |
|        | TS_OFFSET (if TSS = 1)   |            |                |
|        | TIME_STRIDE (if TIS = 1) |            |                |
|        | PKT_ARR_TIME (if TIS = 1)|            |                |
|        | LONGEST_LOSS_EVENT(O)    |            |                |
|        | CLOCK_RESOLUTION(O)      |            |                |
|        | MAX_JITTER(O)            |            |                |
+--------+--------------------------+------------+----------------+
```

   1) ACKED: Indicates whether or not ACK has ever been sent.

   2) PKT_ARR_TIME: The arrival time of the packet that most recently
      decompressed and verified using CRC.

      PRE_SN_V_REF: The sequence number of the packet verified before
      the most recently verified packet.

      CSRC_GEN_ID: The CSRC gen_id of the most recently received packet.

      IPEH_GEN_ID: The IPv6 Extension Header gen_id of the most recently
      received packet.

   3) The remaining elements are as defined in the compressor context.

6.5.3.  List compression: Sliding windows in R-mode and U/O-mode

   In R-mode list compression (see section 5.8.2.1), each entry in the
   sliding window, both at the compressor side and at the decompressor
   side, has the following structure:

```
+---------------------+--------+------------+
| RTP Sequence Number | icount | index list |
+---------------------+--------+------------+
```

The table index list contains a list of index.  Each of these index
corresponds to the item in the original list carried in the packet
identified by the RTP Sequence Number.  The mapping between the index
and the item is identified in the translation table.  The icount
field carries the number of index in the following index list.

In U/O-mode list compression, each entry in the sliding window at
both the compressor side and decompressor side has the following
structure.

```
+--------+--------+------------+
| Gen_id | icount | index list |
+--------+--------+------------+
```

The icount and index list fields are the same as defined in R-mode.
Instead of using the RTP Sequence Number to identify each entry, the
Gen_id is included in the sliding window in U/O-mode.

7.  Security Considerations

   Because encryption eliminates the redundancy that header compression
   schemes try to exploit, there is some inducement to forego encryption
   of headers in order to enable operation over low-bandwidth links.
   However, for those cases where encryption of data (and not headers)
   is sufficient, RTP does specify an alternative encryption method in
   which only the RTP payload is encrypted and the headers are left in
   the clear.  That would still allow header compression to be applied.

   ROHC compression is transparent with regard to the RTP Sequence
   Number and RTP Timestamp fields, so the values of those fields can be
   used as the basis of payload encryption schemes (e.g., for
   computation of an initialization vector).

   A malfunctioning or malicious header compressor could cause the
   header decompressor to reconstitute packets that do not match the
   original packets but still have valid IP, UDP and RTP headers and
   possibly also valid UDP checksums.  Such corruption may be detected
   with end-to-end authentication and integrity mechanisms which will
   not be affected by the compression.  Moreover, this header
   compression scheme uses an internal checksum for verification of
   reconstructed headers.  This reduces the probability of producing
   decompressed headers not matching the original ones without this
   being noticed.

   Denial-of-service attacks are possible if an intruder can introduce
   (for example) bogus STATIC, DYNAMIC or FEEDBACK packets onto the link
   and thereby cause compression efficiency to be reduced.  However, an
   intruder having the ability to inject arbitrary packets at the link
   layer in this manner raises additional security issues that dwarf
   those related to the use of header compression.

8.  IANA Considerations

   The ROHC profile identifier is a non-negative integer. In many
   negotiation protocols, it will be represented as a 16-bit value.  Due
   to the way the profile identifier is abbreviated in ROHC packets, the
   8 least significant bits of the profile identifier have a special
   significance: Two profile identifiers with identical 8 LSBs should be
   assigned only if the higher-numbered one is intended to supersede the
   lower-numbered one.  To highlight this relationship, profile
   identifiers should be given in hexadecimal (as in 0x1234, which would
   for example supersede 0x0A34).

   Following the policies outlined in [IANA-CONSIDERATIONS], the IANA
   policy for assigning new values for the profile identifier shall be
   Specification Required: values and their meanings must be documented
   in an RFC or in some other permanent and readily available reference,
   in sufficient detail that interoperability between independent
   implementations is possible.  In the 8 LSBs, the range 0 to 127 is
   reserved for IETF standard-track specifications; the range 128 to 254
   is available for other specifications that meet this requirement
   (such as Informational RFCs).  The LSB value 255 is reserved for
   future extensibility of the present specification.

   The following profile identifiers are already allocated:

   Profile       Document       Usage
   identifier

   0x0000        RFCthis        ROHC uncompressed
   0x0001        RFCthis        ROHC RTP
   0x0002        RFCthis        ROHC UDP
   0x0003        RFCthis        ROHC ESP

9.  Acknowledgments

   Earlier header compression schemes described in [CJHC], [IPHC], and
   [CRTP] have been important sources of ideas and knowledge.

   The editor would like to extend his warmest thanks to Mikael
   Degermark, who actually did a lot of the editing work, and Peter
   Eriksson, who made a copy editing pass through the document,
   significantly increasing its editorial consistency.  Of course, all
   remaining editorial problems have then been inserted by the editor.

   Thanks to Andreas Jonsson (Lulea University), who supported this work
   by his study of header field change patterns.

   Finally, this work would not have succeeded without the continual
   advice in navigating the IETF standards track, garnished with both
   editorial and technical comments, from the IETF transport area
   directors, Allison Mankin and Scott Bradner.

10.  Intellectual Property Right Claim Considerations

   The IETF has been notified of intellectual property rights claimed in
   regard to some or all of the specification contained in this
   document.  For more information consult the online list of claimed
   rights.

   The IETF takes no position regarding the validity or scope of any
   intellectual property or other rights that might be claimed to
   pertain to the implementation or use of the technology described in
   this document or the extent to which any license under such rights
   might or might not be available; neither does it represent that it
   has made any effort to identify any such rights.  Information on the
   IETF's procedures with respect to rights in standards-track and
   standards-related documentation can be found in BCP-11.  Copies of
   claims of rights made available for publication and any assurances of
   licenses to be made available, or the result of an attempt made to
   obtain a general license or permission for the use of such
   proprietary rights by implementors or users of this specification can
   be obtained from the IETF Secretariat.

   The IETF invites any interested party to bring to its attention any
   copyrights, patents or patent applications, or other proprietary
   rights which may cover technology that may be required to practice
   this standard.  Please address the information to the IETF Executive
   Director.

11.  References

11.1.  Normative References

   [UDP]               Postel, J., "User Datagram Protocol", STD 6,
                       RFC 768, August 1980.

   [IPv4]              Postel, J.,  "Internet Protocol", STD 5, RFC
                       791, September 1981.

   [IPv6]              Deering, S. and R. Hinden, "Internet Protocol,
                       Version 6 (IPv6) Specification", RFC 2460,
                       December 1998.

   [RTP]               Schulzrinne, H., Casner, S., Frederick, R. and
                       V.  Jacobson, "RTP: A Transport Protocol for
                       Real-Time Applications", RFC 1889, January
                       1996.

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

   [ESP]               Kent, S. and R. Atkinson, "IP Encapsulating
                       Security Payload", RFC 2406, November 1998.

   [NULL]              Glenn, R. and S. Kent, "The NULL Encryption
                       Algorithm and Its Use With Ipsec", RFC 2410,
                       November 1998.

   [AH]                Kent, S. and R. Atkinson, "IP Authentication
                       Header", RFC 2402, November 1998.

   [MINE]              Perkins, C., "Minimal Encapsulation within IP",
                       RFC 2004, October 1996.

   [GRE1]              Farinacci, D., Li, T., Hanks, S., Meyer, D. and
                       P. Traina, "Generic Routing Encapsulation
                       (GRE)", RFC 2784, March 2000.

   [GRE2]              Dommety, G., "Key and Sequence Number
                       Extensions to GRE", RFC 2890, August 2000.

   [ASSIGNED]          Reynolds, J. and J. Postel, "Assigned Numbers",
                       STD 2, RFC 1700, October 1994.

11.2.  Informative References

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

   [IPHC]                 Degermark, M., Nordgren, B. and S. Pink, "IP
                          Header Compression", RFC 2507, February 1999.

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

   [CRTPC]                Degermark, M., Hannu, H., Jonsson, L.E.,
                          Svanbro, K., "Evaluation of CRTP Performance
                          over Cellular Radio Networks", IEEE Personal
                          Communication Magazine, Volume 7, number 4, pp.
                          20-25, August 2000.

   [REQ]                  Degermark, M., "Requirements for robust
                          IP/UDP/RTP header compression", RFC 3096, June
                          2001.

   [LLG]                  Svanbro, K., "Lower Layer Guidelines for Robust
                          RTP/UDP/IP Header Compression", Work in
                          Progress.

   [IANA-CONSIDERATIONS] Alvestrand, H. and T. Narten, "Guidelines for
                          Writing an IANA Considerations Section in
                          RFCs", BCP 26, RFC 2434, October 1998.

12.  Authors' Addresses

   Carsten Bormann, Editor
   Universitaet Bremen TZI
   Postfach 330440
   D-28334 Bremen, Germany

   Phone: +49 421 218 7024
   Fax:   +49 421 218 7000
   EMail: cabo@tzi.org


   Carsten Burmeister
   Panasonic European Laboratories GmbH
   Monzastr. 4c
   63225 Langen, Germany

   Phone: +49-6103-766-263
   Fax:   +49-6103-766-166
   EMail: burmeister@panasonic.de


   Mikael Degermark
   The University of Arizona
   Dept of Computer Science
   P.O. Box 210077
   Tucson, AZ 85721-0077, USA

   Phone: +1 520 621-3498
   Fax:   +1 520 621-4642
   EMail: micke@cs.arizona.edu


   Hideaki Fukushima
   Matsushita Electric Industrial Co.,
   Ltd006, Kadoma, Kadoma City,
   Osaka, Japan

   Phone: +81-6-6900-9192
   Fax:   +81-6-6900-9193
   EMail: fukusima@isl.mei.co.jp

Hans Hannu
Box 920
Ericsson Erisoft AB
SE-971 28 Lulea, Sweden

Phone: +46 920 20 21 84
Fax:   +46 920 20 20 99
EMail: hans.hannu@ericsson.com


Lars-Erik Jonsson
Box 920
Ericsson Erisoft AB
SE-971 28 Lulea, Sweden

Phone: +46 920 20 21 07
Fax:   +46 920 20 20 99
EMail: lars-erik.jonsson@ericsson.com


Rolf Hakenberg
Panasonic European Laboratories GmbH
Monzastr. 4c
63225 Langen, Germany

Phone: +49-6103-766-162
Fax:   +49-6103-766-166
EMail: hakenberg@panasonic.de


Tmima Koren
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA  95134, USA

Phone: +1 408-527-6169
EMail: tmima@cisco.com

   Khiem Le
   2-700
   Mobile Networks Laboratory
   Nokia Research Center
   6000 Connection Drive
   Irving, TX 75039, USA

   Phone: +1-972-894-4882
   Fax:   +1 972 894-4589
   EMail: khiem.le@nokia.com


   Zhigang Liu
   2-700
   Mobile Networks Laboratory
   Nokia Research Center
   6000 Connection Drive
   Irving, TX 75039, USA

   Phone: +1 972 894-5935
   Fax:   +1 972 894-4589
   EMail: zhigang.liu@nokia.com


   Anton Martensson
   Ericsson Radio Systems AB
   Torshamnsgatan 23
   SE-164 80 Stockholm, Sweden

   Phone: +46 8 404 3881
   Fax:   +46 8 757 5550
   EMail: anton.martensson@era.ericsson.se


   Akihiro Miyazaki
   Matsushita Electric Industrial Co., Ltd
   1006, Kadoma, Kadoma City, Osaka, Japan

   Phone: +81-6-6900-9192
   Fax:   +81-6-6900-9193
   EMail: akihiro@isl.mei.co.jp

   Krister Svanbro
   Box 920
   Ericsson Erisoft AB
   SE-971 28 Lulea, Sweden

   Phone: +46 920 20 20 77
   Fax:   +46 920 20 20 99
   EMail: krister.svanbro@ericsson.com


   Thomas Wiebke
   Panasonic European Laboratories GmbH
   Monzastr. 4c
   63225 Langen, Germany

   Phone: +49-6103-766-161
   Fax:   +49-6103-766-166
   EMail: wiebke@panasonic.de


   Takeshi Yoshimura
   NTT DoCoMo, Inc.
   3-5, Hikarinooka
   Yokosuka, Kanagawa, 239-8536, Japan

   Phone: +81-468-40-3515
   Fax:   +81-468-40-3788
   EMail: yoshi@spg.yrp.nttdocomo.co.jp


   Haihong Zheng
   2-700
   Mobile Networks Laboratory
   Nokia Research Center
   6000 Connection Drive
   Irving, TX 75039, USA

   Phone: +1 972 894-4232
   Fax:   +1 972 894-4589
   EMail: haihong.zheng@nokia.com

Appendix A.  Detailed classification of header fields

   Header compression is possible thanks to the fact that most header
   fields do not vary randomly from packet to packet.  Many of the
   fields exhibit static behavior or change in a more or less
   predictable way.  When designing a header compression scheme, it is
   of fundamental importance to understand the behavior of the fields in
   detail.

   In this appendix, all IP, UDP and RTP header fields are classified
   and analyzed in two steps.  First, we have a general classification
   in A.1 where the fields are classified on the basis of stable
   knowledge and assumptions.  The general classification does not take
   into account the change characteristics of changing fields because
   those will vary more or less depending on the implementation and on
   the application used.  A less stable but more detailed analysis of
   the change characteristics is then done in A.2.  Finally, A.3
   summarizes this appendix with conclusions about how the various
   header fields should be handled by the header compression scheme to
   optimize compression and functionality.

A.1.  General classification

   At a general level, the header fields are separated into 5 classes:

   INFERRED        These fields contain values that can be inferred from
                   other values, for example the size of the frame
                   carrying the packet, and thus do not have to be
                   handled at all by the compression scheme.

   STATIC          These fields are expected to be constant throughout
                   the lifetime of the packet stream.  Static information
                   must in some way be communicated once.

   STATIC-DEF      STATIC fields whose values define a packet stream.
                   They are in general handled as STATIC.

   STATIC-KNOWN    These STATIC fields are expected to have well-known
                   values and therefore do not need to be communicated
                   at all.

   CHANGING        These fields are expected to vary in some way:
                   randomly, within a limited value set or range, or in
                   some other manner.


   In this section, each of the IP, UDP and RTP header fields is
   assigned to one of these classes.  For all fields except those
   classified as CHANGING, the motives for the classification are also
   stated.  In section A.2, CHANGING fields are further examined and
   classified on the basis of their expected change behavior.

A.1.1.  IPv6 header fields

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

Version

   The version field states which IP version is used.  Packets with
   different values in this field must be handled by different IP
   stacks.  All packets of a packet stream must therefore be of the
   same IP version.  Accordingly, the field is classified as STATIC.

Flow Label

   This field may be used to identify packets belonging to a specific
   packet stream.  If not used, the value should be set to zero.
   Otherwise, all packets belonging to the same stream must have the
   same value in this field, it being one of the fields that define
   the stream.  The field is therefore classified as STATIC-DEF.

Payload Length

   Information about packet length (and, consequently, payload
   length) is expected to be provided by the link layer.  The field
   is therefore classified as INFERRED.

Next Header

   This field will usually have the same value in all packets of a
   packet stream.  It encodes the type of the subsequent header.
   Only when extension headers are sometimes present and sometimes
   not, will the field change its value during the lifetime of the
   stream.  The field is therefore classified as STATIC.

Source and Destination addresses

   These fields are part of the definition of a stream and must thus
   be constant for all packets in the stream.  The fields are
   therefore classified as STATIC-DEF.

Total size of the fields in each class:

```
    +--------------+--------------+
    | Class        | Size (octets)|
    +--------------+--------------+
    | INFERRED     |      2       |
    | STATIC       |      1.5     |
    | STATIC-DEF   |     34.5     |
    | CHANGING     |      2       |
    +--------------+--------------+
```

A.1.2.  IPv4 header fields

```
+---------------------+------------+----------------+
| Field               | Size (bits) |     Class      |
+---------------------+------------+----------------+
| Version             |     4      |     STATIC     |
| 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     |
| Header Checksum     |    16      |    INFERRED    |
| Source Address      |    32      |   STATIC-DEF   |
| Destination Address |    32      |   STATIC-DEF   |
+---------------------+------------+----------------+
```

Version

   The version field states which IP version is used.  Packets with
   different values in this field must be handled by different IP
   stacks.  All packets of a packet stream must therefore be of the
   same IP version.  Accordingly, the field is classified as STATIC.

Header Length

   As long no options are present in the IP header, the header length
   is constant and well known.  If there are options, the fields
   would be STATIC, but it is assumed here that there are no options.
   The field is therefore classified as STATIC-KNOWN.

Packet Length

   Information about packet length is expected to be provided by the
   link layer.  The field is therefore classified as INFERRED.

Flags

   The Reserved flag must be set to zero and is therefore classified
   as STATIC-KNOWN.  The Don't Fragment (DF) flag will be constant
   for all packets in a stream and is therefore classified as STATIC.

Finally, the More Fragments (MF) flag is expected to be zero because fragmentation is NOT expected, due to the small packet size expected.  The More Fragments flag is therefore classified as STATIC-KNOWN.

Fragment Offset

Under the assumption that no fragmentation occurs, the fragment offset is always zero.  The field is therefore classified as STATIC-KNOWN.

Protocol

This field will usually have the same value in all packets of a packet stream.  It encodes the type of the subsequent header. Only when extension headers are sometimes present and sometimes not, will the field change its value during the lifetime of a stream.  The field is therefore classified as STATIC.

Header Checksum

The header checksum protects individual hops from processing a corrupted header. When almost all IP header information is compressed away, there is no point in having this additional checksum; instead it can be regenerated at the decompressor side. The field is therefore classified as INFERRED.

Source and Destination addresses

These fields are part of the definition of a stream and must thus be constant for all packets in the stream.  The fields are therefore classified as STATIC-DEF.

Total size of the fields in each class:

```
+--------------+----------------+
| Class        | Size (octets)  |
+--------------+----------------+
| INFERRED     |       4        |
| STATIC       | 1 oct + 5 bits |
| STATIC-DEF   |       8        |
| STATIC-KNOWN | 2 oct + 3 bits |
| CHANGING     |       4        |
+--------------+----------------+
```

A.1.3.  UDP header fields

```
+-----------------+------------+-------------+
| Field           | Size (bits) |   Class    |
+-----------------+------------+-------------+
| Source Port     |     16     | STATIC-DEF  |
| Destination Port|     16     | STATIC-DEF  |
| Length          |     16     |  INFERRED   |
| Checksum        |     16     |  CHANGING   |
+-----------------+------------+-------------+
```

   Source and Destination ports

      These fields are part of the definition of a stream and must thus
      be constant for all packets in the stream.  The fields are
      therefore classified as STATIC-DEF.

   Length

      This field is redundant and is therefore classified as INFERRED.

   Total size of the fields in each class:

```
+------------+---------------+
| Class      | Size (octets) |
+------------+---------------+
| INFERRED   |       2       |
| STATIC-DEF |       4       |
| CHANGING   |       2       |
+------------+---------------+
```

A.1.4.  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-DEF     |
| CSRC            |   0(-480)  |   CHANGING     |
+-----------------+------------+----------------+
```

Version

   Only one working RTP version exists, namely version 2.  The field
   is therefore classified as STATIC-KNOWN.

Padding

   The use of this field is application-dependent, but when payload
   padding is used it is likely to be present in all packets.  The
   field is therefore classified as STATIC.

Extension

   If RTP extensions are used by the application, these extensions
   are likely to be present in all packets (but the use of extensions
   is very uncommon).  However, for safety's sake this field is
   classified as STATIC and not STATIC-KNOWN.

SSRC

   This field is part of the definition of a stream and must thus be
   constant for all packets in the stream.  The field is therefore
   classified as STATIC-DEF.

Total size of the fields in each class:

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

A.1.5.  Summary for IP/UDP/RTP

   Summarizing this for IP/UDP/RTP one obtains

```
      +----------------+----------------+----------------+
      | Class \ IP ver | IPv6 (octets)  | IPv4 (octets)  |
      +----------------+----------------+----------------+
      | INFERRED       |       4        |       6        |
      | STATIC         | 1 oct + 6 bits | 1 oct + 7 bits |
      | STATIC-DEF     |      42.5      |       16       |
      | STATIC-KNOWN   |     2 bits     | 2 oct + 5 bits |
      | CHANGING       |   11.5(-71.5)  |   13.5(-73.5)  |
      +----------------+----------------+----------------+
      | Total          |   60(-120)     |    40(-100)    |
      +----------------+----------------+----------------+
```

A.2.  Analysis of change patterns of header fields

   To design suitable mechanisms for efficient compression of all header
   fields, their change patterns must be analyzed.  For this reason, an
   extended classification is done based on the general classification
   in A.1, considering the fields which were labeled CHANGING in that
   classification.  Different applications will use the fields in
   different ways, which may affect their behavior.  For the fields
   whose behavior is variable, typical behavior for conversational audio
   and video will be discussed.

   The CHANGING fields are separated into five different subclasses:

   STATIC              These are fields that were classified as
                       CHANGING on a general basis, but are classified
                       as STATIC here due to certain additional
                       assumptions.

   SEMISTATIC          These fields are STATIC most of the time.
                       However, occasionally the value changes but
                       reverts to its original value after a known
                       number of packets.

   RARELY-CHANGING (RC) These are fields that change their values
                       occasionally and then keep their new values.

   ALTERNATING         These fields alternate between a small number
                       of different values.

   IRREGULAR           These, finally, are the fields for which no
                       useful change pattern can be identified.

To further expand the classification possibilities without increasing
complexity, the classification can be done either according to the
values of the field and/or according to the values of the deltas for
the field.

When the classification is done, other details are also stated
regarding possible additional knowledge about the field values and/or
field deltas, according to the classification.  For fields classified
as STATIC or SEMISTATIC, the case could be that the value of the
field is not only STATIC but also well KNOWN a priori (two states for
SEMISTATIC fields).  For fields with non-irregular change behavior,
it could be known that changes usually are within a LIMITED range
compared to the maximal change for the field.  For other fields, the
values are completely UNKNOWN.

Table A.1 classifies all the CHANGING fields on the basis of their
expected change patterns, especially for conversational audio and
video.

| Field | | Value/Delta | Class | Knowledge |
|---|---|---|---|---|
| | Sequential | Delta | STATIC | KNOWN |
| IPv4 Id: | Seq. jump | Delta | RC | LIMITED |
| | Random | Value | IRREGULAR | UNKNOWN |
| IP TOS / Tr. Class | | Value | RC | UNKNOWN |
| IP TTL / Hop Limit | | Value | ALTERNATING | LIMITED |
| UDP Checksum: | Disabled | Value | STATIC | KNOWN |
| | Enabled | Value | IRREGULAR | UNKNOWN |
| RTP CSRC Count: | No mix | Value | STATIC | KNOWN |
| | Mixed | Value | RC | LIMITED |
| RTP Marker | | Value | SEMISTATIC | KNOWN/KNOWN |
| RTP Payload Type | | Value | RC | UNKNOWN |
| RTP Sequence Number | | Delta | STATIC | KNOWN |
| RTP Timestamp | | Delta | RC | LIMITED |
| RTP CSRC List: | No mix | - | - | - |
| | Mixed | Value | RC | UNKNOWN |

Table A.1 : Classification of CHANGING header fields

The following subsections discuss the various header fields in
detail.  Note that table A.1 and the discussions below do not
consider changes caused by loss or reordering before the compression
point.

A.2.1.  IPv4 Identification

   The Identification field (IP ID) of the IPv4 header is there to
   identify which fragments constitute a 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 jump

      This is the most common assignment policy in today's IP stacks.  A
      single IP ID counter is used for all packet streams.  When the
      sender is running more than one packet stream simultaneously, the
      IP ID can increase by more than one between packets in a stream.
      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 and sending frequencies) will usually be limited.

   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.

   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 in the stream, except at wrap around.  Therefore, the
      delta value of the field is constant and well known a priori.
      When RTP is used on top of UDP and IP, the IP ID value follows
      the RTP Sequence Number.  This assignment policy is the most
      desirable for header compression purposes.  However, its usage is
      not as common as it perhaps should be.  The reason may be that it
      can be realized only when UDP and IP are implemented together so
      that UDP, which separates packet streams by the Port
      identification fields, can make IP use separate ID counters for
      each packet stream.

In order to avoid violating [IPv4], packets sharing the same IP
address pair and IP protocol number cannot use the same IP ID
values.  Therefore, implementations of sequential policies must
make the ID number spaces disjoint for packet streams of the same
IP protocol going between the same pair of nodes.  This can be
done in a number of ways, all of which introduce occasional
jumps, and thus makes the policy less than perfectly sequential.
For header compression purposes less frequent jumps are
preferred.

It should be noted that the ID is an IPv4 mechanism and is therefore
not a problem for IPv6.  For IPv4 the ID could be handled in three
different ways.  First, we have the inefficient but reliable solution
where the ID field is sent as-is in all packets, increasing the
compressed headers by two octets.  This is the best way to handle the
ID field if the sender uses random assignment of the ID field.
Second, there can be solutions with more flexible mechanisms
requiring less bits for the ID handling as long as sequential jump
assignment is used.  Such solutions will probably require even more
bits if random assignment is used by the sender.  Knowledge about the
sender's assignment policy could therefore be useful when choosing
between the two solutions above.  Finally, even for IPv4, header
compression could be designed without any additional information for
the ID field included in compressed headers.  To use such schemes, it
must be known which assignment policy for the ID field is being used
by the sender.  That might not be possible to know, which implies
that the applicability of such solutions is very uncertain.  However,
designers of IPv4 stacks for cellular terminals SHOULD use an
assignment policy close to sequential.

A.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.

A.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 due to route changes.

A.2.4.  UDP Checksum

The UDP checksum is optional.  If disabled, its value is constantly
zero and could be compressed away.  If enabled, its value depends on
the payload, which for compression purposes is equivalent to it
changing randomly with every packet.

A.2.5.  RTP CSRC Counter

   This is a counter indicating the number of CSRC items present in the
   CSRC list.  This number is expected to be almost constant on a
   packet- to-packet basis and change by small amounts.  As long as no
   RTP mixer is used, the value of this field is zero.

A.2.6.  RTP Marker

   For audio the marker bit should be set only in the first packet of a
   talkspurt, while for video it should be set in the last packet of
   every picture.  This means that in both cases the RTP marker is
   classified as SEMISTATIC with well-known values for both states.

A.2.7.  RTP Payload Type

   Changes of the RTP payload type within a packet stream are expected
   to be rare.  Applications could adapt to congestion by changing
   payload type and/or frame sizes, but that is not expected to happen
   frequently.

A.2.8.  RTP Sequence Number

   The RTP Sequence Number will be incremented by one for each packet
   sent.

A.2.9.  RTP Timestamp

   In the audio case:

      As long as there are no pauses in the audio stream, the RTP
      Timestamp will be incremented by a constant delta, corresponding
      to the number of samples in the speech frame.  It will thus mostly
      follow the RTP Sequence Number.  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.  However, the
      increment will probably be within a relatively limited range.

   In the video case:

      Between two consecutive packets, the timestamp will either be
      unchanged or increase by a multiple of a fixed value corresponding
      to the picture clock frequency.  The timestamp can also decrease
      by a multiple of the fixed value if B-pictures are used.  The
      delta interval, expressed as a multiple of the picture clock
      frequency, is in most cases very limited.

A.2.10.  RTP Contributing Sources (CSRC)

   The participants in a session, which are identified by the CSRC
   fields, are expected to be almost the same on a packet-to-packet
   basis with relatively few additions and removals.  As long as RTP
   mixers are not used, no CSRC fields are present at all.

A.3.  Header compression strategies

   This section elaborates on what has been done in previous sections.
   On the basis of the classifications, recommendations are given on how
   to handle the various fields in the header compression process.
   Seven different actions are possible; these are listed together with
   the fields to which each action applies.

A.3.1.  Do not send at all

   The fields that have well known values a priori do not have to be
   sent at all.  These are:

   - IPv6 Payload Length
   - IPv4 Header Length
   - IPv4 Reserved Flag
   - IPv4 Last Fragment Flag
   - IPv4 Fragment Offset

   - UDP Checksum (if disabled)
   - RTP Version

A.3.2.  Transmit only initially

   The fields that are constant throughout the lifetime of the packet
   stream have to be transmitted and correctly delivered to the
   decompressor only once.  These are:

   - IP Version
   - IP Source Address
   - IP Destination Address
   - IPv6 Flow Label
   - IPv4 May Fragment Flag
   - UDP Source Port
   - UDP Destination Port
   - RTP Padding Flag
   - RTP Extension Flag
   - RTP SSRC

A.3.3.  Transmit initially, but be prepared to update

   The fields that are changing only occasionally must be transmitted
   initially but there must also be a way to update these fields with
   new values if they change.  These fields are:

   - IPv6 Next Header
   - IPv6 Traffic Class
   - IPv6 Hop Limit
   - IPv4 Protocol
   - IPv4 Type Of Service (TOS)
   - IPv4 Time To Live (TTL)
   - RTP CSRC Counter
   - RTP Payload Type
   - RTP CSRC List

   Since the values of the IPv4 Protocol and the IPv6 Next Header fields
   are in effect linked to the type of the subsequent header, they
   deserve special treatment when subheaders are inserted or removed.

A.3.4.  Be prepared to update or send as-is frequently

   For fields that normally either are constant or have values deducible
   from some other field, but that frequently diverge from that
   behavior, there must be an efficient way to update the field value or
   send it as-is in some packets.  These fields are:

   - IPv4 Identification (if not sequentially assigned)
   - RTP Marker
   - RTP Timestamp

A.3.5.  Guarantee continuous robustness

   For fields that behave like a counter with a fixed delta for ALL
   packets, the only requirement on the transmission encoding is that
   packet losses between compressor and decompressor must be tolerable.
   If several such fields exist, all these can be communicated together.
   Such fields can also be used to interpret the values for fields
   listed in the previous section.  Fields that have this counter
   behavior are:

   - IPv4 Identification (if sequentially assigned)
   - RTP Sequence Number

A.3.6.  Transmit as-is in all packets

   Fields that have completely random values for each packet must be
   included as-is in all compressed headers.  Those fields are:

   - IPv4 Identification (if randomly assigned)
   - UDP Checksum (if enabled)

A.3.7.  Establish and be prepared to update delta

   Finally, there is a field that is usually increasing by a fixed delta
   and is correlated to another field.  For this field it would make
   sense to make that delta part of the context state.  The delta must
   then be initiated and updated in the same way as the fields listed in
   A.3.3.  The field to which this applies is:

   - RTP Timestamp

Full Copyright Statement

Acknowledgement