Audio/Video Transport Working Group
Internet Draft
March 9, 2000
Expires October 2000
draft-koren-avt-crtp-enhance-01.txt

Tmima Koren
Stephen Casner
Patrick Ruddy
Bruce Thompson
Alex Tweedly
Dan Wing
Cisco Systems
John Geevarghese
Motorola India

Enhancements to IP/UDP/RTP Header Compression

This draft is being submitted as a possible work item to the IETF Audio/Video Transport working group. To subscribe to the mailing list send a message to rem-conf-request@es.net with the line "subscribe" in the body of the message. Archives are available from:
ftp://ftp.es.net/pub/mail-archive/rem-conf

Abstract

This document describes enhancements to CRTP, the header compression algorithm for RTP streams described in [RFC2508]. Each enhancement addresses issues with RFC2508 in different deployment scenarios. Each section below provides a description of the proposed enhancement, the scenario where it is useful and the justification for its use.

Each of these enhancements could be evaluated separately.

The enhancements are applicable both for IPv4 and IPv6.

The IPCP option æIP header compressionÆ (described in RFC2509) is also extended to negotiate using the CRTP enhancements.

1.0 Introduction

As IP/UDP/RTP header compression becomes more widely deployed, it is being used in scenarios where a compressed link could extend over a long physical distance and involve multiple layer-2 switching points. An example of such a link is RTP transport over ATM AAL-5, where the "link" would actually traverse through multiple layer-2 switching points on the path from the CRTP transmitter (compressor) to the CRTP receiver (decompressor). Another example is a wireless link. Such links may experience significant packet loss and/or long round trip delays. Contexts get invalidated due to packet loss, but the CRTP error recovery mechanism using CONTEXT_STATE messages is not efficient due to the long round trip delay.

In scenarios such as this, it is desirable to minimize context invalidation. This document suggests several methods of error prevention and recovery. The suggested enhancements make CRTP more robust and resilient to packet loss, which in turn will reduce context invalidation.

1.1 Specification of Requirements

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 [RFC2119].

2. CRTP Enhancements

2.1 The negative cache stream flag

Certain streams, known or suspected to not be RTP, can be placed in a "negative cache" at the compressor, so only the IP and UDP headers are compressed. It is beneficial to notify the decompressor that the compressed stream is in the negative cache: for such streams the context is shorter - there is no need to include the RTP header, and all RTP-related calculations can be avoided.

In this enhancement, a new flag bit "N" is added to the FULL_HEADER packet that initializes a context at the decompressor. The bit occupied by the new flag was previously always set to zero. If the N flag is set to 1, this indicates that no COMPRESSED_RTP packets will be transmitted in this context. This flag is only an optimization and the decompressor may choose to ignore it.

Format of the FULL_HEADER length fields with the negative cache flag:

For 8-bit context ID:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1| Generation|      CID      |  First length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            0        |N|  seq  |  Second length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  N=1: negative cache stream
```

For 16-bit context ID:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|1| Generation|   0 |N|  seq  |  First length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  N=1: negative cache stream

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              CID              |  Second length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2.2 Reject a new compressed stream

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. In an end-to-end scheme a host may have compressed sessions with many hosts and eventually may run out of resources. When the end-to-end tunnel is negotiated, the number of contexts needed may not be predictable. This enhancement allows the negotiated number of contexts to be larger than could be accommodated if many tunnels are established. Then, as context resources are consumed, an attempt to set up a new context may be rejected.
The compressor initiates a compression of a stream by sending a FULL_HEADER packet. Currently if the decompressor has insufficient resources to decompress the new stream, it can send a CONTEXT_STATE packet to invalidate the newly compressed stream. The compressor does not know the reason for the invalidation: usually this happens when the decompressor gets out of synchronization due to packet loss. The compressor will most likely reattempt to compress this stream by sending another FULL_HEADER.
This enhancement specifies that the decompressor may reject the compression of a stream by sending a REJECT message to the compressor. A REJECT message tells the compressor to stop compressing this stream. The REJECT message is a CONTEXT_STATE message with an additional flag:

```
   Type code = 1 :CONTEXT_STATE for 8-bit CID streams
   Type code = 2 :  CONTEXT_STATE for16-bit CID streams
```

Here is the format of CONTEXT_STATE packets with REJECT flags:

```
                 0   1   2   3   4   5   6   7
               +---+---+---+---+---+---+---+---+
               |Type code=1: CS, 8-bit CID |
               +---+---+---+---+---+---+---+---+
               |         context count         |
               +---+---+---+---+---+---+---+---+
               |       session context ID      |
               +---+---+---+---+---+---+---+---+
               | 1 |R=1| 0 | 0 |    sequence   |   R is the REJECT flag
               +---+---+---+---+---+---+---+---+
               | 0 | 0 |       generation      |
               +---+---+---+---+---+---+---+---+
                              . . .
               +---+---+---+---+---+---+---+---+
               |       session context ID      |
               +---+---+---+---+---+---+---+---+
               | 1 |R=1| 0 | 0 |    sequence   |   R is the REJECT flag
               +---+---+---+---+---+---+---+---+
               | 0 | 0 |       generation      |
               +---+---+---+---+---+---+---+---+



                 0   1   2   3   4   5   6   7
               +---+---+---+---+---+---+---+---+
               |Type code=2: CS, 16-bit CID|
               +---+---+---+---+---+---+---+---+
               |         context count         |
               +---+---+---+---+---+---+---+---+
               |                               |
               +       session context ID      +
               |                               |
               +---+---+---+---+---+---+---+---+
               | 1 |R=1| 0 | 0 |    sequence   |   R is the REJECT flag
               +---+---+---+---+---+---+---+---+
               | 0 | 0 |       generation      |
               +---+---+---+---+---+---+---+---+
                              . . .
               +---+---+---+---+---+---+---+---+
               |                               |
               +       session context ID      +
               |                               |
               +---+---+---+---+---+---+---+---+
               | 1 |R=1| 0 | 0 |    sequence   |   R is the REJECT flag
               +---+---+---+---+---+---+---+---+
               | 0 | 0 |       generation      |
               +---+---+---+---+---+---+---+---+
```

The session CID, sequence and generation are taken from the

FULL_HEADER.

The compressor may decide to wait for a while before attempting to compress additional streams destined to the rejecting host.

2.3 Including IP ID in the UDP checksum

A UDP checksum can be used by the decompressor to verify validity of the packet it reconstructed, especially when the 'twice' algorithm is used. When the ætwiceÆ algorithm was defined in RFC 2507 and subsequently incorporated into RFC 2508, the fact that the IP ID field is not included in the checksum was overlooked. Since the IP ID field is conveyed with a delta value, accurate reconstruction of the IP ID field cannot be verified using the current specifications.

This enhancement modifies the function of the UDP checksum to include the IP ID value, but only between the compressor and decompressor. That is, when a UDP checksum is present (nonzero), the compressor will 1Æs complement subtract the IP ID value from the UDP checksum before compression and the decompressor will 1Æs complement add the IP ID value to the UDP checksum after any validation operations and before delivering the packet further downstream.

2.4 CRTP Headers Checksum

When a UDP checksum is not present (has value zero) in a stream, the compressor MAY replace it with a 16-bit headers checksum (HDRCKSUM). The HDRCKSUM can be used to validate the IP ID and all the headers in the reconstructed packet. Hence it can be used by the decompressor to validate reconstructed packets when ætwiceÆ is used, and to validate every 16Æth packet as recommended in RFC2508, Section 3.3.5.

A new flag in the FULL_HEADER packet, as specified below, indicates when set that all COMPRESSED_UDP and COMPRESSED_RTP packets sent in that context will have HDRCKSUM inserted. If a packet in the same stream subsequently arrives at the compressor with a UDP checksum present, then a new FULL_HEADER packet must be sent with the flag cleared to re-establish the context.

The HDRCKSUM is calculated in the same way as a UDP checksum, but includes only the pseudo-IP header (as defined for UDP), the IP ID (as in Section 2.3), the UDP header and for COMPRESSED_RTP packets, the fixed part of the RTP header (first 12 bytes). The extended part of the RTP header and the RTP data will not be included in the HDRCKSUM. The HDRCKSUM is placed in the COMPRESSED_UDP or COMPRESSED_RTP packets where a UDP checksum would have been.
The decompressor MUST zero out the UDP checksum field in the

reconstructed packets.

The HDRCKSUM does not validate the RTP data. If the link layer is configured to deliver packets without checking for errors, errors in the RTP data will not be detected. Over such links, the compressor SHOULD add the HDRCKSUM if a UDP checksum is not present, and the decompressor SHOULD validate each reconstructed packet to make sure that at least the headers are correct. This ensures that the packet will be delivered to the right destination. If only HDRCKSUM is available, the RTP data will be delivered even if it includes errors. This might be a desirable feature for applications that can tolerate errors in the RTP data. Same holds for the extended part of the RTP header.

Here is the format of the FULL_HEADER length fields with the new flag that indicates that a header checksum will be added in COMPRESSED_UDP and COMPRESSED_RTP packets:

For 8-bit context ID:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1| Generation|      CID      |  First length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            0      |C|N|  seq  |  Second length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  C=1: HDRCKSUM will be added
```

For 16-bit context ID:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|1| Generation| 0 |C|N|  seq  |  First length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  C=1: HDRCKSUM will be added

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              CID              |  Second length field
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2.5 Enhancement to COMPRESSED_UDP packet format (CU*)

The COMPRESSED_UDP packet includes the whole RTP header, so it can restore all RTP-related parameters at the decompressor. It is also specified to reset the delta RTP timestamp to zero and the delta RTP sequence number to zero.It can also convey a new value for the delta IP ID.

It is possible to accommodate some packet loss between the compressor and decompressor using the "twice" algorithm in RFC 2508, but this requires reliably communicating the absolute values and the deltas for the differential fields. The reliability of communication of the absolute values in the RTP header can be increased by sending a COMPRESSED_UDP packet repeatedly, but this resets the delta timestamp. RFC 2508 describes the format of COMPRESSED_UDP as being the same as COMPRESSED_RTP except that the M, S and T bits are always 0 and the corresponding delta fields are never included. This enhancement changes that specification to say that the T bit may be nonzero to indicate that the RTP timestamp delta is included explicitly rather than being reset to zero.

Sometimes it is necessary to change just a few fields of the RTP header. A second part of this enhancement adds more flag bits to the COMPRESSED_UDP packet to select individual uncompressed fields of the RTP header to be included in the packet. Since there are flag bits to indicate inclusion of both delta values and absolute values, the flag nomenclature is changed. The original S,T,I bits which indicate the inclusion of deltas are renamed dS, dT, dI, and the inclusion of absolute values is indicated by S,T,I. The M bit is absolute as before.

The format of the flags/sequence byte for the original COMPRESSED_UDP packet is shown here for reference:

```
            +---+---+---+---+---+---+---+---+
            | 0 | 0 | 0 |dI | link sequence |
            +---+---+---+---+---+---+---+---+
```

The new definition of the flags/sequence byte plus an extension flags byte is as follows, where the new F flag indicates the inclusion of the extension flags byte:

```
            +---+---+---+---+---+---+---+---+
            | F | I |dT |dI | link sequence |
            +---+---+---+---+---+---+---+---+
            : M : S : T :pt :      CC       :  (if F = 1)
            +...+...+...+...+...............+
```

```
dI = delta IP ID
dT = delta RTP timestamp
I  = absolute IP ID
F  = additional flags byte
M  = marker bit
S  = absolute RTP sequence number
T  = absolute RTP timestamp
pt = RTP payload type
CC = number of CSRC identifiers
```

Some short notations:

FH	FULL_HEADER
CR	COMPRESSED_RTP
CR+	COMPRESSED_RTP with delta fields
CU	COMPRESSED_UDP
CU*	enhanced COMPRESSED_UDP

When F=0, there is only one flags byte, and the only available flags
are: dI, dT and I.
In this case the packet includes the full RTP header
If dT=0, the decompressor sets deltaT to 0
If dI=0, the decompressor sets deltaI to 1

Some example packet formats will illustrate the use of the new flags.
First, a 'traditional' COMPRESSED_UDP with full RTP header, when F=0:

```
             0   1   2   3   4   5   6   7
           +...............................+
           :   msb of session context ID   :  (if 16-bit CID)
           +-------------------------------+
           |   lsb of session context ID   |
           +---+---+---+---+---+---+---+---+
           |F=0| I |dT |dI | link sequence |
           +---+---+---+---+---+---+---+---+
           :                               :
           +         UDP checksum          +  (if nonzero in context)
           :                               :
           +...............................+
           :                               :
           +        "RANDOM" fields        +  (if encapsulated)
           :                               :
           +...............................+
           :         delta IPv4 ID         :  (if dI = 1)
           +...............................+
           :      delta RTP timestamp      :  (if dT = 1)
           +...............................+
           :                               :
           +           IPv4 ID             +  (if I = 1)
           :                               :
           +...............................+
           |           UDP data            |
           :   (uncompressed RTP header)   :
```

When F=1, there is an additional flags byte and the available flags
are: dI, dT, I, M, S, T, pt, CC. In this case the packet does not
include the full RTP header, but includes selected fields from the RTP

```
header as specified by the flags. The delta values of the context are
not reset even if they are not specified in the packet:
If dT=0, the decompressor KEEPS THE CURRENT deltaT
   (and DOES NOT set the deltaT to 0)
If dI=0, the decompressor KEEPS THE CURRENT deltaI
   (and DOES NOT set the the deltaI to 1)


A CU* packet is similar in contents and behavior to a CR packet, but it
has more flag bits, some of which correspond to absolute values for RTP
header fields.


COMPRESSED_UDP with individual RTP fields, when F=1 :

              0   1   2   3   4   5   6   7
             +...............................+
             :   msb of session context ID   :  (if 16-bit CID)
             +-------------------------------+
             |   lsb of session context ID   |
             +---+---+---+---+---+---+---+---+
             |F=1| I |dT |dI | link sequence |
             +---+---+---+---+---+---+---+---+
             : M : S : T :pt :      CC       :
             +...+...+...+...+...............+
             :                               :
             +         UDP checksum          +  (if nonzero in context)
             :                               :
             +...............................+
             :                               :
             :        "RANDOM" fields        :  (if encapsulated)
             :                               :
             +...............................+
             :         delta IPv4 ID         :  (if dI = 1)
             +...............................+
             :      delta RTP timestamp      :  (if dT = 1)
             +...............................+
             :                               :
             +           IPv4 ID             +  (if I = 1)
             :                               :
             +...............................+
             :                               :
             +     RTP sequence number       +  (if S = 1)
             :                               :
             +...............................+
             :                               :
             +                               +
             :                               :
             +         RTP timestamp         +  (if T = 1)
             :                               :
             +                               +
             :                              :
             +..............................+
             :       RTP payload type       :  (if pt = 1)
             +..............................+
             :                              :
             :           CSRC list          :  (if CC > 0)
             :                              :
             +..............................+
             :                              :
             :      RTP header extension    :  (if X set in context)
             :                              :
             +------------------------------+
             |                              |
             /           RTP data           /
             /                              /
             |                              |
             +------------------------------+
             :            padding           :  (if P set in context)
             +..............................+
```

Usage for the CU* packet:

It is useful for the compressor to periodically refresh the state of the decompressor to avoid having the decompressor send CONTEXT_STATE messages in the case of unrecoverable packet loss.
Using the flags F=0 I dI dT, this CU* packet refreshes all the context parameters.

When compression is done over a lossy link with a long round trip delay, we want to minimize context invalidation. If the delta values are changing frequently, the context might get invalidated often. In such cases the compressor may choose to include absolute values in the CRTP packets instead of delta values, using CU* packets with the flags: F=1, and any of S, T, I as necessary.

2.6 Acknowledgement packet (ACK packet)

The ACK packet will be sent from decompressor to compressor to indicate receipt of a compressed packet with the ACK'd RTP sequence number.
ItÆs a CONTEXT_STATE packet with type codes 4 and 5. The ACK packet is to be used in a separately negotiated mode of operation as described in the next section.

```
Type code = 4 :  ACK a packet of a context with 8-bit CID
Type code = 5 :  ACK a packet of a context with 16-bit CID
```

The format for the ACK packet is:

```
             0   1   2   3   4   5   6   7
```

```
+---+---+---+---+---+---+---+---+
|  Type code=4: ACK, 8-bit CID  |
+---+---+---+---+---+---+---+---+
|          context count        |
+---+---+---+---+---+---+---+---+
|        session context ID     |
+---+---+---+---+---+---+---+---+

|                               |
+      RTP sequence number      +
|                               |
+---+---+---+---+---+---+---+---+
              . . .
+---+---+---+---+---+---+---+---+
|        session context ID     |
+---+---+---+---+---+---+---+---+

|                               |
+      RTP sequence number      +
|                               |
+---+---+---+---+---+---+---+---+


  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|  Type code=5: ACK, 16-bit CID |
+---+---+---+---+---+---+---+---+
|          context count        |
+---+---+---+---+---+---+---+---+
|                               |
+       session context ID      +
|                               |
+---+---+---+---+---+---+---+---+

|                               |
+      RTP sequence number      +
|                               |
+---+---+---+---+---+---+---+---+
              . . .
+---+---+---+---+---+---+---+---+
|                               |
+       session context ID      +
|                               |
+---+---+---+---+---+---+---+---+

|                               |
+      RTP sequence number      +
|                               |
+---+---+---+---+---+---+---+---+
```

2.6.1 **CRTP operation in ACK mode**

This mode of operation is optional and must be negotiated per link.

2.6.1.1 **Description of the ACK mode**

The ACK mode is a mode of operation in which the compressor and decompressor continuously verify that their context states are synchronized. The compressor repeatedly notifies the decompressor about changes in the context state, until the decompressor acknowledges reception of the changes by sending ACK packets to the decompressor. This effort of synchronizing the context states helps minimize context invalidation.

The context state shared between the compressor and decompressor includes all the fields of the uncompressed headers and the first order differences (delta fields) of the fields that change by a constant value from packet to packet. Each field follows its known change pattern: either stays constant or is incremented by its corresponding delta field. Fields that follow their change pattern are compressed. They are reconstructed by the decompresor from the context state at the decompressor. Correct decompression of a packet depends on whether the context state at the compressor when the packet is compressed and sent is identical to the context state at the decompressor when that packet is received and decompressed.

When a field changes in a way that is different from its change pattern, the compressor assigns a new value to the field, and stores it in the context state at the compressor side. The decompressor must be informed about the change so that it can update the state on its side to match the state at the compressor. The compressor notifies the decompressor about such changes by including information about the changed field in the compressed packet. (for example if dT was assigned a new value, the compressor can send a CR+ packet that includes dT). The context is not synchronized until the decompressor receives the packet that includes the changed field and updates its state accordingly.

The decompressor indicates reception of the change by sending an ACK packet to the compressor. The ACK packet includes the RTP sequence number of the packet that it is ACKÆing, so the compressor can identify which packet is ACKÆd. The compressor canÆt assume that the decompressor received the change until the ACK packet is received.

Depending on the round trip delay of the link, the compressor might have to send a few more packets before the ACK from the decompressor arrives. In this case the compressor must repeat the change in all subsequent packets. Reception of the ACK is an indication that the decompressor updated its context with the changed value. Now that their contexts are synchronized again, the compressor can stop including the changed field in the compressed packets.

The decompressor must be able to recognize the repeat packets (the packets that repeat the same change and were sent while the compressor was waiting for the ACK packet). Those repeat packets donÆt require an ACK.

If in the process of changing some fields additional changes are required, the compressor will switch to send packets that include all changes. The decompressor must ACK one of the packets that include all the changes.

The compressor and decompressor must be in full agreement about which packets must be ACKÆd: packets that include changes are larger in size, and if they are not ACKÆd, the changes are repeated in all subsequent packets, and bandwidth is wasted.

LetÆs summarize which packets require an ACK:

1. **A Packet that assigns a value to any context state field** must be ACKÆd. This includes FH and CU packets because they initialize fields in the context state.
2. **Repeat packets donÆt require an ACK**

How are repeat packets identified?

A packet is considered to be a repeat packet if:
1. **It updates the same fields as the previous packet**
2. **Each field is updated by a value that is equal to the one assigned** to this field in the previous packet plus the corresponding delta for this field, when applicable.

2.6.1.2 The Random IP ID

The IP ID change pattern is to be incremented by dI. In some implementations, the IP ID counter is shared across multiple streams, so as a result of the varying mix of packets the increment for any particular stream is not constant. When compressing such a stream, the compressor must include in each packet either dI or I. It is recommended to include I rather then dI because a loss of a packet that includes a new delta value dI will invalidate the context.
According to the rules set above, each packet will have to be ACKÆd.

To correct this weÆll define a new change pattern for the IP ID: random value. The IP ID assumes this change pattern when dI is set to be 0.

We add a rule to the ACK rules:
3. **When the value of dI is 0, packets that update only the IP ID field**

donÆt require an ACK.

And add to rule 2 of the repeat packet rules:
2. **Each field is updated by a value that is equal to the one assigned** to this field in the previous packet plus the corresponding delta for this field, when applicable. An exception to this rule is the IP ID field: if the value of dI is 0, the IP ID may be assigned any value.

2.6.1.3 Implementation hints when using the ACK scheme

1. **When a delta field is updated, add the matching absolute field too** (dT and T, dI and I). Loss of a packet that updates only the delta value can easily cause context invalidation.
2. **Set dI=0 when the IP ID is changing randomly, and include I in all** packets.
3. **If you ACKÆd a packet, but the number of repeat packets exceed your** estimate, ACK again (your previous ACK was probably lost)

Here is an example to demonstrate the usage of the ACK scheme.
In this stream the audio codec sends a sample every 10 msec
The first talk spurt is 1 second long. Then there are 2 seconds silence, then another talk spurt.

When there is no loss on the link, we can use the following sequence:
(The deltaID is not constant so we send deltaID in each packet)

```
seq#  Time  pkt type
1     10    FH
2     20    CR+     dI dT=10
3     30    CR+     dI
4     40    CR+     dI
...
100   1000  CR+     dI

101   3010  CR+     dI dT=2010
102   3020  CR+     dI dT=10
103   3030  CR+     dI
104   3040  CR+     dI
...
```

In the above sequence if a packet is lost, we cannot recover ('twice' will not work due to the random IP ID) and the context must be invalidated.

Here is the same sequence using the ACK scheme(CU* is the enhanced CU):

```
seq#  Time  pkt type  flags
1     10    FH                                   FH must be ACK'd
```

```
2     20    FH                                        repeat
ACK 1
3     30    CU*   1 I dT dI M 0 T 0   I T=30 dT=10 dI=0  dI,dT changed
 (packet 3 was lost)                                 (I and T sent too)
4     40    CU*   1 I dT dI M 0 T 0   I T=40 dT=10 dI=0  repeat
5     50    CU*   1 I dT dI M 0 T 0   I T=50 dT=10 dI=0  repeat
6     60    CU*   1 I dT dI M 0 T 0   I T=60 dT=10 dI=0  repeat
ACK 4 == got new dI=0 and dT=10 at T=40.
          dI was set to 0, so I does not require an ACK.
          No need to ACK 5 and 6: repeat packets
7     70    CU*   1 I  0 0  M 0 0 0   I
8     80    CU*   1 I  0 0  M 0 0 0   I
...
100   1000  CU*   1 I  0 0  M 0 0 0   I

101   3010  CU*   1 I  0 0  M 0 T 0   I T=3010   T changed, keep
deltas!
102   3020  CU*   1 I  0 0  M 0 T 0   I T=3020   repeat
ACK 101 == got new T at sequence 101
           No need to ACK packet 102 because 3010 + dT = 3020
           If 101 is lost, 102 will be ACK'd
103   3030  CU*   1 I  0 0  M 0 0 0   I
104   3040  CU*   1 I  0 0  M 0 0 0   I
...


The same sequences, when delta IP ID is constant:

seq#  Time  pkt type
1     10    FH
2     20    CR+     dI dT=10
3     30    CR
4     40    CR
...
100   1000  CR

101   3010  CR+    dT=2010
102   3020  CR+    dT=10
103   3030  CR
104   3040  CR
...



seq#  Time  pkt type  flags
1     10    FH                                        FH must be ACK'd
2     20    FH                                        repeat
ACK 1
3     30    CU*   1 I dT dI M 0 T 0   I dI T=30 dT=10  dI,dT changed
  (packet 3 was lost)                                   (I and T sent
too)
4     40    CU*   1 I dT dI M 0 T 0   I dI T=40 dT=10  repeat
5     50    CU*   1 I dT dI M 0 T 0   I dI T=50 dT=10  repeat
6     60    CU*   1 I dT dI M 0 T 0   I dI T=60 dT=10  repeat
ACK 4 == got new dI and dT=10 at T=40.
          No need to ACK 5 and 6: no changes
7     70    CR
8     80    CR
...
100   1000  CR

101   3010  CU*   1 0  0 0  M 0 T 0   T=3010    T changed, keep deltas!
102   3020  CU*   1 0  0 0  M 0 T 0   T=3020    repeat
ACK 101 == got new T at sequence 101
          No need to ACK packet 102 because 3010 + dT = 3020
          If 101 is lost, 102 will be ACK'd
103   3030  CR
104   3040  CR
...
```

2.8 CRTP operation in 'N' mode

This scheme is similar to the ACK scheme in that the compressor tries to keep the decompressor in sync by sending changes multiple times. The 'N' is a number that represents the quality of the link between the hosts, and it means that the probability of more than 'N' adjacent packets getting lost on this link is small. For every change in a base value or a delta value, if the compressor includes the change in N+1 consecutive packets, there is a very good chance that the compressor and decompressor can stay in sync using the 'twice' algorithm. CONTEXT_STATE packets should also be repeated N+1 times (using the same sequence number).
It is up to the implementation to find a scheme to derive an appropriate N for a link.

This scheme may be used at any time and does not require negotiation.

Here is the same example in 'N' mode, when N=2 and deltaID is constant:

```
seq#  Time  pkt type  flags
1     10    FH
2     20    FH                                repeat constant fields
3     30    FH                                repeat constant fields
4     40    CU*   1 I dT dI M 0 T 0   I dI T=40 dT=10
5     50    CU*   1 I dT dI M 0 T 0   I dI T=50 dT=10  repeat delta
6     60    CU*   1 I dT dI M 0 T 0   I dI T=60 dT=10  repeat delta
7     70    CR
8     80    CR
```

```
...
100   1000  CR

101   3010  CU*   1 0  0 0  M 0 T 0   T=3010    T changed, keep deltas!
102   3020  CU*   1 0  0 0  M 0 T 0   T=3020    repeat updated T
103   3030  CU*   1 0  0 0  M 0 T 0   T=3030    repeat updated T
104   3040  CR
105   3050  CR
...
```

2.9 Negotiating usage of enhanced-CRTP and ACK scheme

RFC 2509 [IPCPHP] specifies how the use of CRTP is negotiated on PPP links using the IP Compression Protocol option of IPCP:

```
    IPCP option 2: IP compression protocol
    protocol 0x61 indicates RFC 2507 header compression
    sub-option 1 enables use of COMPRESSED_RTP, COMPRESSED_UDP and
                  CONTEXT_STATE as specified in RFC 2508
```

For the enhancements defined in this document, two new sub-options are added:

```
    sub-option 2 (length=2) :  enables use of CRTP with
                               enhancements 2.1 - 2.5
    sub-option 3 (length=2) :  enables use of CRTP with
                               enhancements 2.1 - 2.6 (ACK scheme)
```

3. Acknowledgements

The authors would like to thank Van Jacobson, co-author of RFC2508, and the authors of RFC2507, Mikael Degermark, Bjorn Nordgren, and Stephen Pink. The authors would also like to thank Dana Blair, Francois Le Faucheur, Tim Gleeson, Matt Madison, Hussein Salama, Mallik Tatipamula, Mike Thomas, and Herb Wildfeuer.

4. References

[CRTP] S. Casner, V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC2508, February 1999.

[IPHCOMP] M. Degermark, B. Nordgren, S. Pink, "IP Header Compression", RFC2507, February 1999.

[IPCPHC] M. Engan, S. Casner, C. Bormann, "IP Header Compression over PPP", RFC2509, February 1999.

[KEYW] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC2119, BCP 14, March 1997.

[RTP] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC1889, January 1996.

5. Authors' Addresses

Stephen L. Casner
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
United States of America

Email: casner@cisco.com

John Geevarghese
Motorola India Electronics Ltd,
Ulsoor Road
Bangalore - 42
India

Email: gvjohn@miel.mot.com

Tmima Koren
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
United States of America
Email: tmima@cisco.com

Patrick Ruddy
Cisco Systems, Inc.
3rd Floor, 96 Commercial Street
Edinburgh
EH6 6LX
Scotland

Email: pruddy@cisco.com

Bruce Thompson
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
United States of America

Email: brucet@cisco.com

Alex Tweedly
Cisco Systems, Inc.
3 The Square, Stockley Park
Uxbridge, Middlesex
UB11 1BN
United Kingdom

Email: agt@cisco.com

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
United States of America

Email: dwing@cisco.com

6. Copyright