Network Working Group
Request for Comments: 3545
Category: Standards Track

T. Koren
Cisco Systems
S. Casner
Packet Design
J. Geevarghese
Motorola India Electronics Ltd.
B. Thompson
P. Ruddy
Cisco Systems

July 2003

Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering

Status of this Memo

Copyright Notice

Abstract

This document describes a header compression scheme for point to point links with packet loss and long delays. It is based on Compressed Real-time Transport Protocol (CRTP), the IP/UDP/RTP header compression described in RFC 2508. CRTP does not perform well on such links: packet loss results in context corruption and due to the long delay, many more packets are discarded before the context is repaired. To correct the behavior of CRTP over such links, a few extensions to the protocol are specified here. The extensions aim to reduce context corruption by changing the way the compressor updates the context at the decompressor: updates are repeated and include updates to full and differential context parameters. With these extensions, CRTP performs well over links with packet loss, packet reordering and long delays.

Table of Contents

1. Introduction

RTP header compression (CRTP) as described in RFC 2508 was designed to reduce the header overhead of IP/UDP/RTP datagrams by compressing the three headers. The IP/UDP/RTP headers are compressed to 2-4 bytes most of the time.

CRTP was designed for reliable point to point links with short delays. It does not perform well over links with high rate of packet loss, packet reordering and long delays.

An example of such a link is a PPP session that is tunneled using an IP level tunneling protocol such as L2TP. Packets within the tunnel are carried by an IP network and hence may get lost and reordered. The longer the tunnel, the longer the round trip time.

Another example is an IP network that uses layer 2 technologies such as ATM and Frame Relay for the access portion of the network. Layer 2 transport networks such as ATM and Frame Relay behave like point to point serial links in that they do not reorder packets. In addition, Frame Relay and ATM virtual circuits used as IP access technologies often have a low bit rate associated with them. These virtual circuits differ from low speed serial links in that they may span a larger physical distance than a point to point serial link. Speed of light delays within the layer 2 transport network will result in higher round trip delays between the endpoints of the circuit. In

addition, congestion within the layer 2 transport network may result in an effective drop rate for the virtual circuit which is significantly higher than error rates typically experienced on point to point serial links.

It may be desirable to extend existing CRTP implementations for use also over IP tunnels and other virtual circuits, where packet losses, reordering, and long delays are common characteristics. To address these scenarios, this document defines modifications and extensions to CRTP to increase robustness to both packet loss and misordering between the compressor and the decompressor. This is achieved by repeating updates and allowing the sending of absolute (uncompressed) values in addition to delta values for selected context parameters. Although these new mechanisms impose some additional overhead, the overall compression is still substantial. The enhanced CRTP, as defined in this document, is thus suitable for many applications in the scenarios discussed above, e.g., tunneling and other virtual circuits.

RFC 3095 defines another RTP header compression scheme called Robust Header Compression [ROHC]. ROHC was developed with wireless links as the main target, and introduced new compression mechanisms with the primary objective to achieve the combination of robustness against packet loss and maximal compression efficiency. ROHC is expected to be the preferred compression mechanism over links where compression efficiency is important. However, ROHC was designed with the same link assumptions as CRTP, e.g., that the compression scheme should not have to tolerate misordering of compressed packets between the compressor and decompressor, which may occur when packets are carried in an IP tunnel across multiple hops.

At some time in the future, enhancements may be defined for ROHC to allow it to perform well in the presence of misordering of compressed packets. The result might be more efficient than the compression protocol specified in this document. However, there are many environments for which the enhanced CRTP defined here may be the preferred choice. In particular, for those environments where CRTP is already implemented, the additional effort required to implement the extensions defined here is expected to be small. There are also cases where the implementation simplicity of this enhanced CRTP relative to ROHC is more important than the performance advantages of ROHC.

1.1. CRTP Operation

During compression of an RTP stream, a session context is defined. For each context, the session state is established and shared between the compressor and the decompressor. Once the context state is established, compressed packets may be sent.

The context state consists of the full IP/UDP/RTP headers, a few first order differential values, a link sequence number, a generation number and a delta encoding table.

The headers part of the context is set by the FULL_HEADER packet that always starts a compression session. The first order differential values (delta values) are set by sending COMPRESSED_RTP packets that include updates to the delta values.

The context state must be synchronized between compressor and decompressor for successful decompression to take place. If the context gets out of sync, the decompressor is not able to restore the compressed headers accurately. The decompressor invalidates the context and sends a CONTEXT_STATE packet to the compressor indicating that the context has been corrupted. To resume compression, the compressor must re-establish the context.

During the time the context is corrupted, the decompressor discards all the packets received for that context. Since the context repair mechanism in CRTP involves feedback from the decompressor, context repair takes at least as much time as the round trip time of the link. If the round trip time of the link is long, and especially if the link bandwidth is high, many packets will be discarded before the context is repaired. On such links it is desirable to minimize context invalidation.

1.2. How do contexts get corrupted?

As long as the fields in the combined IP/UDP/RTP headers change as expected for the sequence of packets in a session, those headers can be compressed, and the decompressor can fully restore the compressed headers using the context state. When the headers don’t change as expected it’s necessary to update some of the full or the delta values of the context. For example, the RTP timestamp is expected to increment by delta RTP timestamp (dT). If silence suppression is used, packets are not sent during silence periods. Then when voice activity resumes, packets are sent again, but the RTP timestamp is incremented by a large value and not by dT. In this case an update must be sent.

If a packet that includes an update to some context state values is lost, the state at the decompressor is not updated. The shared state is now different at the compressor and decompressor. When the next packet arrives at the decompressor, the decompressor will fail to restore the compressed headers accurately since the context state at the decompressor is different than the state at the compressor.

1.3. Preventing context corruption

Note that the decompressor fails not when a packet is lost, but when the next compressed packet arrives. If the next packet happens to include the same context update as in the lost packet, the context at the decompressor may be updated successfully and decompression may continue uninterrupted. If the lost packet included an update to a delta field such as the delta RTP timestamp (dT), the next packet can't compensate for the loss since the update of a delta value is relative to the previous packet which was lost. But if the update is for an absolute value such as the full RTP timestamp or the RTP payload type, this update can be repeated in the next packet independently of the lost packet. Hence it is useful to be able to update the absolute values of the context.

The next chapter describes several extensions to CRTP that add the capability to selectively update absolute values of the context, rather than sending a FULL_HEADER packet, in addition to the existing updates of the delta values. This enhanced version of CRTP is intended to minimize context invalidation and thus improve the performance over lossy links with a long round trip time.

1.4. 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. Enhanced CRTP

This chapter specifies the changes in this enhanced version of CRTP. They are:

- Extensions to the COMPRESSED_UDP packet to allow updating the differential RTP values in the decompressor context and to selectively update the absolute IPv4 ID and the following RTP values: sequence number, timestamp, payload type, CSRC count and CSRC list. This allows context sync to be maintained even with some packet loss.

- A "headers checksum" to be inserted by the compressor and removed by the decompressor when the UDP checksum is not present so that validation of the decompressed headers is still possible. This allows the decompressor to verify that context sync has not been lost after a packet loss.

An algorithm is then described to use these changes with repeated updates to achieve robust operation over links with packet loss and long delay.

2.1. Extended COMPRESSED_UDP packet

It is possible to accommodate some packet loss between the compressor and decompressor using the "twice" algorithm in RFC 2508 so long as the context remains in sync. In that algorithm, the delta values are added to the previous context twice (or more) to effect the change that would have occurred if the missing packets had arrived. The result is verified with the UDP checksum. Keeping the context in sync requires reliably communicating both the absolute value and the delta value whenever the delta value changes. For many environments, sufficient reliability can be achieved by repeating the update with each of several successive packets.

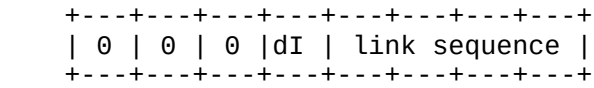
The COMPRESSED_UDP packet satisfies the need to communicate the absolute values of the differential RTP fields, but it is specified in RFC 2508 to reset the delta RTP timestamp. That limitation can be removed with the following simple change: RFC 2508 describes the format of COMPRESSED_UDP as being the same as COMPRESSED_RTP except that the M, S and T bits are always 0 and the corresponding delta fields are never included. This enhanced version of CRTP changes that specification to say that the T bit MAY be nonzero to indicate that the delta RTP timestamp is included explicitly rather than being reset to zero.

A second change adds another byte of flag bits to the COMPRESSED_UDP packet to allow only selected individual uncompressed fields of the RTP header to be included in the packet rather than carrying the full RTP header as part of the UDP data. The additional flags do increase computational complexity somewhat, but the corresponding increase in bit efficiency is important when the differential field updates are communicated multiple times in successive COMPRESSED_UDP packets. With this change, there are flag bits to indicate inclusion of both delta values and absolute values, so 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. A new

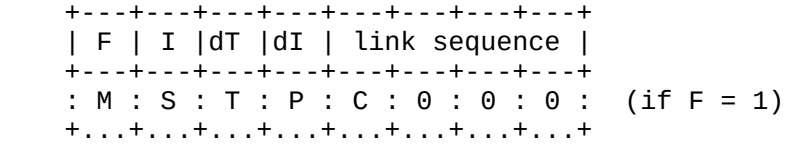
flag P indicates inclusion of the absolute RTP payload type value and another flag C indicates the inclusion of the CSRC count. When C=1, an additional byte is added following the two flag bytes to include the absolute value of the four-bit CC field in the RTP header.

The last of the three changes to the COMPRESSED_UDP packet deals with updating the IPv4 ID field. For this field, the COMPRESSED_UDP packet as specified in RFC 2508 can already convey a new value for the delta IPv4 ID, but not the absolute value which is only conveyed by the FULL_HEADER packet. Therefore, a new flag I is added to the COMPRESSED_UDP packet to indicate inclusion of the absolute IPv4 ID value. The I flag replaces the dS flag which is not needed in the COMPRESSED_UDP packet since the delta RTP sequence number always remains 1 in the decompressor context and hence does not need to be updated. Note that IPv6 does not have an IP ID field, so when compressing IPv6 packets both the I and the dI flags are always set to 0.

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



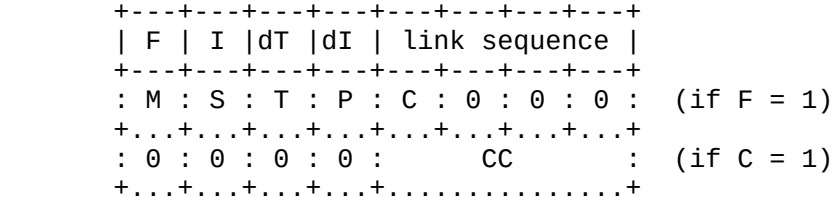
The new definition of the flags/sequence byte plus an extension flags byte for the COMPRESSED_UDP packet is as follows, where the new F flag indicates the inclusion of the extension flags byte:



dI = delta IPv4 ID
dT = delta RTP timestamp
I = absolute IPv4 ID
F = additional flags byte
M = marker bit
S = absolute RTP sequence number
T = absolute RTP timestamp
P = RTP payload type
C = CSRC count
CID = Context ID

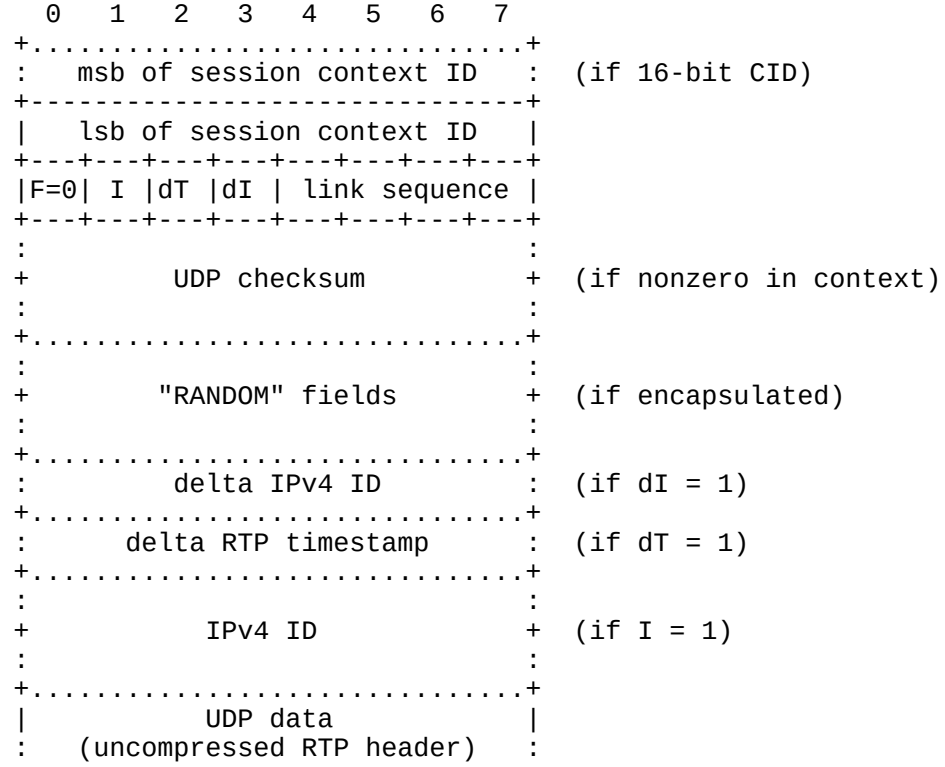
When F=0, there is only one flags byte, and the only available flags are: dI, dT and I. In this case the packet includes the full RTP header. As in RFC 2508, if dI=0, the decompressor does not change deltaI. If dT=0, the decompressor sets deltaT to 0.

When C=1, an additional byte is added following the two flag bytes. This byte includes the CC, the count of CSRC identifiers, in its lower 4 bits:



The bits marked "0" in the second flag byte and the CC byte SHOULD be set to zero by the sender and SHOULD be ignored by the receiver.

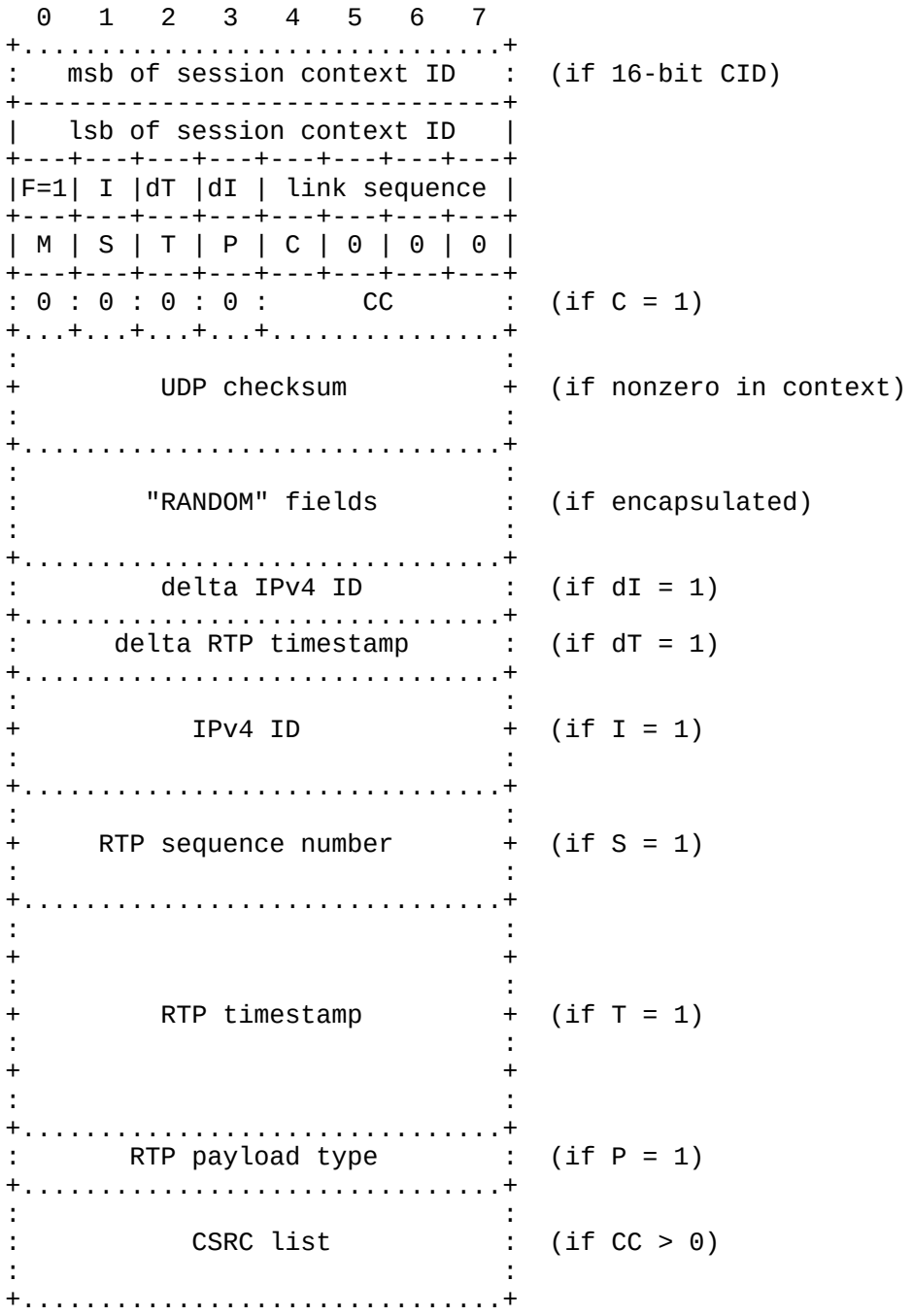
Some example packet formats will illustrate the use of the new flags. First, when F=0, the "traditional" COMPRESSED_UDP packet which carries the full RTP header as part of the UDP data:

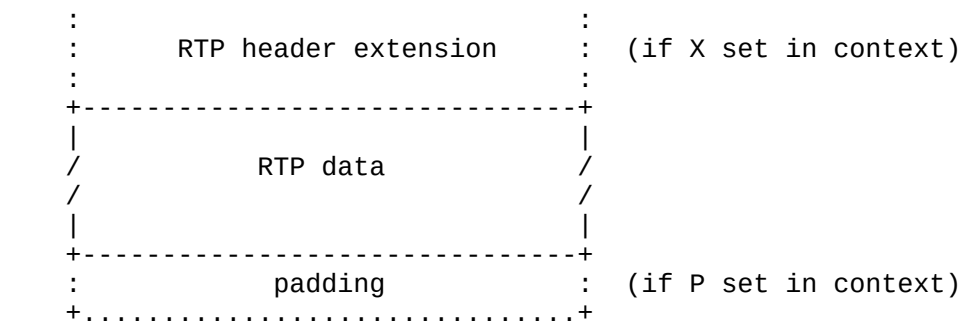


When F=1, there is an additional flags byte and the available flags are: dI, dT, I, M, S, T, P, C. If C=1, there is an additional byte that includes the number of CSRC identifiers. When F=1, the packet does not include the full RTP header, but includes selected fields from the RTP header as specified by the flags. As in RFC 2508, if dI=0 the decompressor does not change deltaI. However, in contrast to RFC 2508, if dT=0 the decompressor KEEPS THE CURRENT deltaT in the context (DOES NOT set deltaT to 0).

An enhanced COMPRESSED_UDP packet is similar in contents and behavior to a COMPRESSED_RTP 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:





Usage for the enhanced COMPRESSED_UDP 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 and I=1, dI=1, dT=1, the COMPRESSED_UDP 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 always send absolute values and never delta values, using COMPRESSED_UDP packets with the flags F=1, and any of S, T, I as necessary.

2.2. CRTP Headers Checksum

RFC 2508, in Section 3.3.5, describes how the UDP checksum may be used to validate header reconstruction periodically or when the "twice" algorithm is used. When a UDP checksum is not present (has value zero) in a stream, such validation would not be possible. To cover that case, this enhanced CRTP provides an option whereby the compressor MAY replace the null UDP checksum with a 16-bit headers checksum (HDRCKSUM) which is subsequently removed by the decompressor after validation. Note that this option is never used with IPv6 since a null UDP checksum is not allowed.

A new flag C 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. The compressor MAY set the C flag when UDP packet carried in the FULL_HEADER packet originally contained a checksum value of zero. If the C flag is set, the FULL_HEADER packet itself MUST also have the 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.

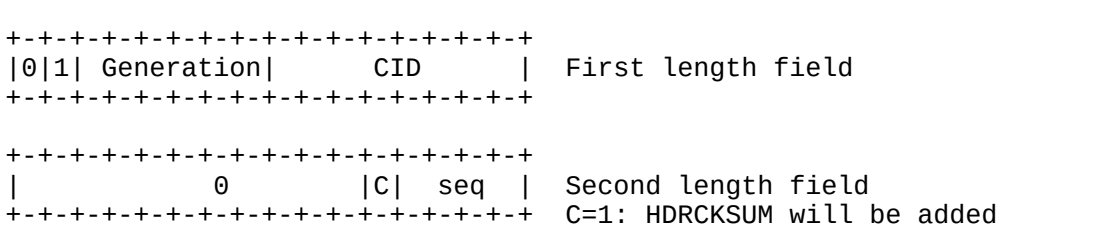
The HDRCKSUM is calculated in the same way as a UDP checksum except that it does not cover all of the UDP data. That is, the HDRCKSUM is the 16-bit one's complement of the one's complement sum of the pseudo-IP header (as defined for UDP), the UDP header, the first 12 bytes of the UDP data which are assumed to hold the fixed part of an RTP header, and the CSRC list. The extended part of the RTP header beyond the CSRC list and the RTP data will not be included in the HDRCKSUM. The HDRCKSUM is placed in the COMPRESSED_UDP or COMPRESSED_RTP packet where a UDP checksum would have been. The decompressor MUST zero out the UDP checksum field in the reconstructed packets.

For a non-RTP context, there may be fewer than 12 UDP data bytes present. The IP and UDP headers can still be compressed into a COMPRESSED_UDP packet. For this case, the HDRCKSUM is calculated over the pseudo-IP header, the UDP header, and the UDP data bytes that are present. If the number of data bytes is odd, then a zero padding byte is appended for the purpose of calculating the checksum, but not transmitted.

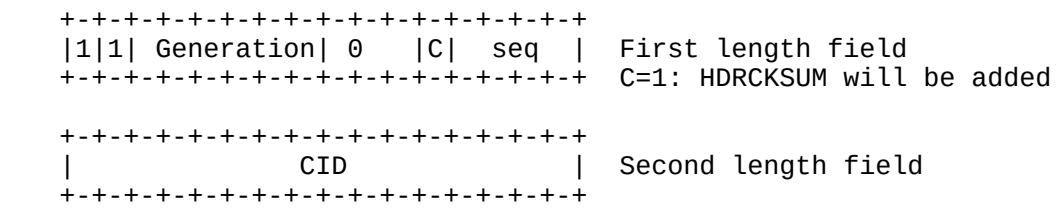
The HDRCKSUM does not validate the RTP data. If the link layer is configured to deliver packets without checking for errors, then errors in the RTP data will not be detected. Over such links, the compressor SHOULD add the HDRCKSUM if a UDP checksum is not present, and the decompressor SHOULD validate 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. The same holds for the extended part of the RTP header beyond the CSRC list.

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

For 8-bit context ID:



For 16-bit context ID:



2.3. Achieving robust operation

Enhanced CRTP achieves robust operation by sending changes multiple times to keep the compressor and decompressor in sync. This method is characterized by a number "N" that represents the quality of the link between the hosts. What it means is that the probability of more than N adjacent packets getting lost on this link is small. For every change in a full value or a delta value, if the compressor includes the change in N+1 consecutive packets, then the decompressor can keep its context state in sync with the compressor using the "twice" algorithm so long as no more than N adjacent packets are lost.

Since updates are repeated in N+1 packets, if at least one of these N+1 update packets is received by the decompressor, both the full and delta values in the context at the decompressor will get updated and its context will stay synchronized with the context at the compressor. We can conclude that as long as less than N+1 adjacent packets are lost, the context at the decompressor is guaranteed to be synchronized with the context at the compressor, and use of the "twice" algorithm to recover from packet loss will successfully update the context and restore the compressed packets.

The link sequence number cycles in 16 packets, so it’s not always clear how many packets were lost. For example, if the previous link sequence number was 5 and the current number is 4, one possibility is that 15 packets were lost, but another possibility is that due to misordering packet 5 arrived before packet 4 and they are really adjacent. If there is an interpretation of the link sequence numbers that could be a gap of less than N+1, the "twice" algorithm may be applied that many times and verified with the UDP checksum (or the HDRCKSUM).

When more than N packets are lost, all of the repetitions of an update might have been lost. The context state may then be different at the compressor and decompressor. The decompressor can still try to recover by making one or more guesses for how many packets were lost and then applying the "twice" algorithm that many times.

However, since the IPv4 ID field is not included in the checksum, this does not validate the IPv4 ID.

The conclusion is that for IPv4 if more than N packets were lost, the decompressor SHOULD NOT try to recover using the "twice" algorithm and instead SHOULD invalidate the context and send a CONTEXT_STATE packet. In IPv6 the decompressor MAY always try to recover from packet loss by using the "twice" algorithm and verifying the result with the UDP checksum.

It is up to the implementation to derive an appropriate N for a link. The value is maintained independently for each context and is not required to be the same for all contexts. When compressing a new stream, the compressor sets a value of N for that context and sends N+1 FULL_HEADER packets. The compressor MUST also repeat each subsequent COMPRESSED_UDP update N+1 times. The value of N may be changed for an existing context by sending a new sequence of FULL_HEADER packets.

The decompressor learns the value of N by counting the number of times the FULL_HEADER packet is repeated and storing the resulting value in the corresponding context. If some of the FULL_HEADER packets are lost, the decompressor may still be able to determine the correct value of N by observing the change in the 4-bit sequence number carried in the FULL_HEADER packets. Any inaccuracy in the counting will lead the decompressor to assume a smaller value of N than the compressor is sending. This is safe in that the only negative consequence is that the decompressor might send a CONTEXT_STATE packet when it was not really necessary to do so. In response, the compressor will send FULL_HEADER packets again, providing another opportunity for the decompressor to count the correct N.

The sending of FULL_HEADER packets is also triggered by a change in one of the fields held constant in the context, such as the IP TOS. If such a change should occur while the compressor is in the middle of sending the N+1 FULL_HEADER packets, then the compressor MUST send N+1 FULL_HEADER packets after making the change. This could cause the decompressor to receive more than N+1 FULL_HEADER packets in a row with the result that it assumes a larger value for N than is correct. That could lead to an undetected loss of context synchronization. Therefore, the compressor MUST change the "generation" number in the context and in the FULL_HEADER packet when it begins sending the sequence of N+1 FULL_HEADER packets so the decompressor can detect the new sequence. For IPv4, this is a change in behavior relative to RFC 2508.

CONTEXT_STATE packets SHOULD also be repeated N+1 times (using the same sequence number for each context) to provide a similar measure of robustness against packet loss. Here N can be the largest N of all contexts included in the CONTEXT_STATE packet, or any number the decompressor finds necessary in order to ensure robustness.

2.3.1. Examples

Here are some examples to demonstrate the robust operation of enhanced CRTP using N+1 repetitions of updates. In this stream the audio codec sends a sample every 10 milliseconds. The first talkspurt is 1 second long. Then there are 2 seconds of silence, then another talkspurt. We also assume in this first example that the IPv4 ID field does not increment at a constant rate because the host is generating other uncorrelated traffic streams at the same time and therefore the delta IPv4 ID changes for each packet.

In these examples, we will use some short notations:

```
 FH    FULL_HEADER
 CR    COMPRESSED_RTP
 CU    COMPRESSED_UDP
```

When operating on a link with low loss, we can just use COMPRESSED_RTP packets in the basic CRTP method specified in RFC 2508. We might have the following packet sequence:

```
 seq Time pkt    updates and comments
  #       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 unpredictable IPv4 ID) and the context must be invalidated.

Here is the same example using the enhanced CRTP method specified in this document, when N=2. Note that the compressor only sends the absolute IPv4 ID (I) and not the delta IPv4 ID (dI).

```
 seq Time pkt  CU flags            updates and comments
  #       type F I dT dI M S T P
 1   10   FH
 2   20   FH                           repeat constant fields
 3   30   FH                           repeat constant fields
 4   40   CU   1 1  1  0 M 0 1 0   I T=40 dT=10
 5   50   CU   1 1  1  0 M 0 1 0   I T=50 dT=10 repeat update T & dT
 6   60   CU   1 1  1  0 M 0 1 0   I T=60 dT=10 repeat update T & dT
 7   70   CU   1 1  0  0 M 0 0 0   I
 8   80   CU   1 1  0  0 M 0 0 0   I
 ...
 100 1000 CU   1 1  0  0 M 0 0 0   I

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

This second example is the same sequence, but assuming the delta IP ID is constant. First the basic CRTP for a lossless link:

```
 seq Time pkt    updates and comments
  #       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
 ...
```

For the equivalent sequence in enhanced CRTP, the more efficient COMPRESSED_RTP packet can still be used once the deltas are all established:

```
 seq Time pkt  CU flags            updates and comments
  #       type F I dT dI M S T P
 1   10   FH
 2   20   FH                           repeat constant fields
 3   30   FH                           repeat constant fields
 4   40   CU   1 1  1  1 M 0 1 0   I dI T=40 dT=10
 5   50   CU   1 1  1  1 M 0 1 0   I dI T=50 dT=10  repeat updates
 6   60   CU   1 1  1  1 M 0 1 0   I dI T=60 dT=10  repeat updates
 7   70   CR
 8   80   CR
 ...
 100 1000 CR

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

Here is the second example when using IPv6. First the basic CRTP for a lossless link:

```
 seq Time pkt    updates and comments
  #       type
 1   10   FH
 2   20   CR     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
 ...
```

For the equivalent sequence in enhanced CRTP, the more efficient COMPRESSED_RTP packet can still be used once the deltas are all established:

```
 seq Time pkt  CU flags            updates and comments
  #       type F I dT dI M S T P
 1   10   FH
 2   20   FH                           repeat constant fields
 3   30   FH                           repeat constant fields
 4   40   CU   1 0  1  0 M 0 1 0   T=40 dT=10
 5   50   CU   1 0  1  0 M 0 1 0   T=50 dT=10  repeat updates
 6   60   CU   1 0  1  0 M 0 1 0   T=60 dT=10  repeat updates
 7   70   CR
 8   80   CR
 ...
 100 1000 CR

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

3. Negotiating usage of enhanced-CRTP

The use of IP/UDP/RTP compression (CRTP) over a particular link is a function of the link-layer protocol. It is expected that negotiation of the use of CRTP will be defined separately for each link layer.

For link layers that already have defined a negotiation for the use of CRTP as specified in RFC 2508, an extension to that negotiation will be required to indicate use of the enhanced CRTP defined in this document since the syntax of the existing packet formats has been extended.

4. Security Considerations

Because encryption eliminates the redundancy that this compression scheme tries to exploit, there is some inducement to forego encryption in order to achieve operation over a low-bandwidth link. However, for those cases where encryption of data and not headers is satisfactory, RTP does specify an alternative encryption method in which only the RTP payload is encrypted and the headers are left in the clear [SRTP]. That would allow compression to still be applied.

A malfunctioning or malicious compressor could cause the decompressor to reconstitute packets that do not match the original packets but still have valid IP, UDP and RTP headers and possibly even valid UDP check-sums. Such corruption may be detected with end-to-end authentication and integrity mechanisms which will not be affected by the compression. Constant portions of authentication headers will be compressed as described in [IPHCOMP].

No authentication is performed on the CONTEXT_STATE control packet sent by this protocol. An attacker with access to the link between the decompressor and compressor could inject false CONTEXT_STATE packets and cause compression efficiency to be reduced, probably resulting in congestion on the link. However, an attacker with access to the link could also disrupt the traffic in many other ways.

A potential denial-of-service threat exists when using compression techniques that have non-uniform receiver-end computational load. The attacker can inject pathological datagrams into the stream which are complex to decompress and cause the receiver to be overloaded and degrading processing of other streams. However, this compression does not exhibit any significant non-uniformity.

5. Acknowledgements

The authors would like to thank Van Jacobson, co-author of RFC 2508, and the authors of RFC 2507, 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, Alex Tweedly, Herb Wildfeuer, Andrew Johnson, and Dan Wing.

6. References

6.1. Normative References

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

[IPHCOMP] Degermark, M., Nordgren, B. and S. Pink, "IP Header Compression", RFC 2507, February 1999.

[IPCPHC] Koren, T., Casner, S. and C. Bormann, "IP Header Compression over PPP", RFC 3544, July 2003.

[KEYW] Bradner, S. "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RTP] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

6.2. Informative References

[ROHC] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T. and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, July 2001.

[SRTP] Baugher, M., McGrew, D., Carrara, E., Naslund, M. and K. Norrman, "The Secure Real-time Transport Protocol", Work in Progress.

7. Intellectual Property Rights Notice

8. Authors' Addresses

Tmima Koren
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA

EMail: tmima@cisco.com

Stephen L. Casner
Packet Design
3400 Hillview Avenue, Building 3
Palo Alto, CA 94304
USA

EMail: casner@acm.org

John Geevarghese
Motorola India Electronics Ltd.
No. 33 A Ulsoor Road
Bangalore, India

EMail: geevjohn@hotmail.com

Bruce Thompson
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA

EMail: brucet@cisco.com

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

EMail: pruddy@cisco.com

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.