Network Working Group
Request for Comments: 2508
Category: Standards Track

S. Casner
Cisco Systems
V. Jacobson
Cisco Systems

February 1999

Compressing IP/UDP/RTP Headers for Low-Speed Serial Links

Status of this Memo

Copyright Notice

Abstract

This document describes a method for compressing the headers of IP/UDP/RTP datagrams to reduce overhead on low-speed serial links. In many cases, all three headers can be compressed to 2-4 bytes.

Comments are solicited and should be addressed to the working group mailing list rem-conf@es.net and/or the author(s).

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.

1. Introduction

Since the Real-time Transport Protocol was published as an RFC [1], there has been growing interest in using RTP as one step to achieve interoperability among different implementations of network audio/video applications. However, there is also concern that the 12-byte RTP header is too large an overhead for 20-byte payloads when operating over low speed lines such as dial-up modems at 14.4 or 28.8 kb/s. (Some existing applications operating in this environment use an application-specific protocol with a header of a few bytes that has reduced functionality relative to RTP.)

Header size may be reduced through compression techniques as has been done with great success for TCP [2]. In this case, compression might be applied to the RTP header alone, on an end-to-end basis, or to the combination of IP, UDP and RTP headers on a link-by-link basis. Compressing the 40 bytes of combined headers together provides substantially more gain than compressing 12 bytes of RTP header alone because the resulting size is approximately the same (2-4 bytes) in either case. Compressing on a link-by-link basis also provides better performance because the delay and loss rate are lower. Therefore, the method defined here is for combined compression of IP, UDP and RTP headers on a link-by-link basis.

This document defines a compression scheme that may be used with IPv4, IPv6 or packets encapsulated with more than one IP header, though the initial focus is on IPv4. The IP/UDP/RTP compression defined here is intended to fit within the more general compression framework specified in [3] for use with both IPv6 and IPv4. That framework defines TCP and non-TCP as two classes of transport above IP. This specification creates IP/UDP/RTP as a third class extracted from the non-TCP class.

2. Assumptions and Tradeoffs

The goal of this compression scheme is to reduce the IP/UDP/RTP headers to two bytes for most packets in the case where no UDP checksums are being sent, or four bytes with checksums. It is motivated primarily by the specific problem of sending audio and video over 14.4 and 28.8 dialup modems. These links tend to provide full-duplex communication, so the protocol takes advantage of that fact, though the protocol may also be used with reduced performance on simplex links. This compression scheme performs best on local links with low round-trip-time.

This specification does not address segmentation and preemption of large packets to reduce the delay across the slow link experienced by small real-time packets, except to identify in Section 4 some interactions between segmentation and compression that may occur. Segmentation schemes may be defined separately and used in conjunction with the compression defined here.

It should be noted that implementation simplicity is an important factor to consider in evaluating a compression scheme. Communications servers may need to support compression over perhaps as many as 100 dial-up modem lines using a single processor. Therefore, it may be appropriate to make some simplifications in the design at the expense of generality, or to produce a flexible design that is general but can be subsetted for simplicity. Higher compression gain might be achieved by communicating more complex

models for the changing header fields from the compressor to the decompressor, but that complexity is deemed unnecessary. The next sections discuss some of the tradeoffs listed here.

2.1. Simplex vs. Full Duplex

In the absence of other constraints, a compression scheme that worked over simplex links would be preferred over one that did not. However, operation over a simplex link requires periodic refreshes with an uncompressed packet header to restore compression state in case of error. If an explicit error signal can be returned instead, the delay to recovery may be shortened substantially. The overhead in the no-error case is also reduced. To gain these performance improvements, this specification includes an explicit error indication sent on the reverse path.

On a simplex link, it would be possible to use a periodic refresh instead. Whenever the decompressor detected an error in a particular packet stream, it would simply discard all packets in that stream until an uncompressed header was received for that stream, and then resume decompression. The penalty would be the potentially large number of packets discarded. The periodic refresh method described in Section 3.3 of [3] applies to IP/UDP/RTP compression on simplex links or links with high delay as well as to other non-TCP packet streams.

2.2. Segmentation and Layering

Delay induced by the time required to send a large packet over the slow link is not a problem for one-way audio, for example, because the receiver can adapt to the variance in delay. However, for interactive conversations, minimizing the end-to-end delay is critical. Segmentation of large, non-real-time packets to allow small real-time packets to be transmitted between segments can reduce the delay.

This specification deals only with compression and assumes segmentation, if included, will be handled as a separate layer. It would be inappropriate to integrate segmentation and compression in such a way that the compression could not be used by itself in situations where segmentation was deemed unnecessary or impractical. Similarly, one would like to avoid any requirements for a reservation protocol. The compression scheme can be applied locally on the two ends of a link independent of any other mechanisms except for the requirements that the link layer provide some packet type codes, a packet length indication, and good error detection.

Conversely, separately compressing the IP/UDP and RTP layers loses too much of the compression gain that is possible by treating them together. Crossing these protocol layer boundaries is appropriate because the same function is being applied across all layers.

3. The Compression Algorithm

The compression algorithm defined in this document draws heavily upon the design of TCP/IP header compression as described in RFC 1144 [2]. Readers are referred to that RFC for more information on the underlying motivations and general principles of header compression.

3.1. The basic idea

In TCP header compression, the first factor-of-two reduction in data rate comes from the observation that half of the bytes in the IP and TCP headers remain constant over the life of the connection. After sending the uncompressed header once, these fields may be elided from the compressed headers that follow. The remaining compression comes from differential coding on the changing fields to reduce their size, and from eliminating the changing fields entirely for common cases by calculating the changes from the length of the packet. This length is indicated by the link-level protocol.

For RTP header compression, some of the same techniques may be applied. However, the big gain comes from the observation that although several fields change in every packet, the difference from packet to packet is often constant and therefore the second-order difference is zero. By maintaining both the uncompressed header and the first-order differences in the session state shared between the compressor and decompressor, all that must be communicated is an indication that the second-order difference was zero. In that case, the decompressor can reconstruct the original header without any loss of information simply by adding the first-order differences to the saved uncompressed header as each compressed packet is received.

Just as TCP/IP header compression maintains shared state for multiple simultaneous TCP connections, this IP/UDP/RTP compression SHOULD maintain state for multiple session contexts. A session context is defined by the combination of the IP source and destination addresses, the UDP source and destination ports, and the RTP SSRC field. A compressor implementation might use a hash function on these fields to index a table of stored session contexts. The compressed packet carries a small integer, called the session context identifier or CID, to indicate in which session context that packet should be interpreted. The decompressor can use the CID to index its table of stored session contexts directly.

Because the RTP compression is lossless, it may be applied to any UDP traffic that benefits from it. Most likely, the only packets that will benefit are RTP packets, but it is acceptable to use heuristics to determine whether or not the packet is an RTP packet because no harm is done if the heuristic gives the wrong answer. This does require executing the compression algorithm for all UDP packets, or at least those with even port numbers (see section 3.4).

Most compressor implementations will need to maintain a "negative cache" of packet streams that have failed to compress as RTP packets for some number of attempts in order to avoid further attempts. Failing to compress means that some fields in the potential RTP header that are expected to remain constant most of the time, such as the payload type field, keep changing. Even if the other such fields remain constant, a packet stream with a constantly changing SSRC field SHOULD be entered in the negative cache to avoid consuming all of the available session contexts. The negative cache is indexed by the source and destination IP address and UDP port pairs but not the RTP SSRC field since the latter may be changing. When RTP compression fails, the IP and UDP headers MAY still be compressed.

Fragmented IP Packets that are not initial fragments and packets that are not long enough to contain a complete UDP header MUST NOT be sent as FULL_HEADER packets. Furthermore, packets that do not additionally contain at least 12 bytes of UDP data MUST NOT be used to establish RTP context. If such a packet is sent as a FULL_HEADER packet, it MAY be followed by COMPRESSED_UDP packets but MUST NOT be followed by COMPRESSED_RTP packets.

3.2. Header Compression for RTP Data Packets

In the IPv4 header, only the total length, packet ID, and header check-sum fields will normally change. The total length is redundant with the length provided by the link layer, and since this compression scheme must depend upon the link layer to provide good error detection (e.g., PPP's CRC [4]), the header checksum may also be elided. This leaves only the packet ID, which, assuming no IP fragmentation, would not need to be communicated. However, in order to maintain lossless compression, changes in the packet ID will be transmitted. The packet ID usually increments by one or a small number for each packet. (Some systems increment the ID with the bytes swapped, which results in slightly less compression.) In the IPv6 base header, there is no packet ID nor header checksum and only the payload length field changes.

In the UDP header, the length field is redundant with the IP total length field and the length indicated by the link layer. The UDP check-sum field will be a constant zero if the source elects not to generate UDP checksums. Otherwise, the checksum must be communicated intact in order to preserve the lossless compression. Maintaining end-to-end error detection for applications that require it is an important principle.

In the RTP header, the SSRC identifier is constant in a given context since that is part of what identifies the particular context. For most packets, only the sequence number and the timestamp will change from packet to packet. If packets are not lost or misordered upstream from the compressor, the sequence number will increment by one for each packet. For audio packets of constant duration, the timestamp will increment by the number of sample periods conveyed in each packet. For video, the timestamp will change on the first packet of each frame, but then stay constant for any additional packets in the frame. If each video frame occupies only one packet, but the video frames are generated at a constant rate, then again the change in the timestamp from frame to frame is constant. Note that in each of these cases the second-order difference of the sequence number and timestamp fields is zero, so the next packet header can be constructed from the previous packet header by adding the first-order differences for these fields that are stored in the session context along with the previous uncompressed header. When the second-order difference is not zero, the magnitude of the change is usually much smaller than the full number of bits in the field, so the size can be reduced by encoding the new first-order difference and transmitting it rather than the absolute value.

The M bit will be set on the first packet of an audio talkspurt and the last packet of a video frame. If it were treated as a constant field such that each change required sending the full RTP header, this would reduce the compression significantly. Therefore, one bit in the compressed header will carry the M bit explicitly.

If the packets are flowing through an RTP mixer, most commonly for audio, then the CSRC list and CC count will also change. However, the CSRC list will typically remain constant during a talkspurt or longer, so it need be sent only when it changes.

3.3. The protocol

The compression protocol must maintain a collection of shared information in a consistent state between the compressor and decompressor. There is a separate session context for each IP/UDP/RTP packet stream, as defined by a particular combination of the IP source and destination addresses, UDP source and destination

ports, and the RTP SSRC field. The number of session contexts to be maintained MAY be negotiated between the compressor and decompressor. Each context is identified by an 8- or 16-bit identifier, depending upon the number of contexts negotiated, so the maximum number is 65536. Both uncompressed and compressed packets MUST carry the context ID and a 4-bit sequence number used to detect packet loss between the compressor and decompressor. Each context has its own separate sequence number space so that a single packet loss need only invalidate one context.

The shared information in each context consists of the following items:

- The full IP, UDP and RTP headers, possibly including a CSRC list, for the last packet sent by the compressor or reconstructed by the decompressor.

- The first-order difference for the IPv4 ID field, initialized to 1 whenever an uncompressed IP header for this context is received and updated each time a delta IPv4 ID field is received in a compressed packet.

- The first-order difference for the RTP timestamp field, initialized to 0 whenever an uncompressed packet for this context is received and updated each time a delta RTP timestamp field is received in a compressed packet.

- The last value of the 4-bit sequence number, which is used to detect packet loss between the compressor and decompressor.

- The current generation number for non-differential coding of UDP packets with IPv6 (see [3]). For IPv4, the generation number may be set to zero if the COMPRESSED_NON_TCP packet type, defined below, is never used.

- A context-specific delta encoding table (see section 3.3.4) may optionally be negotiated for each context.

In order to communicate packets in the various uncompressed and compressed forms, this protocol depends upon the link layer being able to provide an indication of four new packet formats in addition to the normal IPv4 and IPv6 packet formats:

FULL_HEADER - communicates the uncompressed IP header plus any following headers and data to establish the uncompressed header state in the decompressor for a particular context. The FULL-HEADER packet also carries the 8- or 16-bit session context identifier and the 4-bit sequence number to establish

synchronization between the compressor and decompressor. The format is shown in section 3.3.1.

COMPRESSED_UDP - communicates the IP and UDP headers compressed to 6 or fewer bytes (often 2 if UDP checksums are disabled), followed by any subsequent headers (possibly RTP) in uncompressed form, plus data. This packet type is used when there are differences in the usually constant fields of the (potential) RTP header. The RTP header includes a potentially changed value of the SSRC field, so this packet may redefine the session context. The format is shown in section 3.3.3.

COMPRESSED_RTP - indicates that the RTP header is compressed along with the IP and UDP headers. The size of this header may still be just two bytes, or more if differences must be communicated. This packet type is used when the second-order difference (at least in the usually constant fields) is zero. It includes delta encodings for those fields that have changed by other than the expected amount to establish the first-order differences after an uncompressed RTP header is sent and whenever they change. The format is shown in section 3.3.2.

CONTEXT_STATE - indicates a special packet sent from the decompressor to the compressor to communicate a list of context IDs for which synchronization has or may have been lost. This packet is only sent across the point-to-point link so it requires no IP header. The format is shown in section 3.3.5.

When this compression scheme is used with IPv6 as part of the general header compression framework specified in [3], another packet type MAY be used:

COMPRESSED_NON_TCP - communicates the compressed IP and UDP headers as defined in [3] without differential encoding. If it were used for IPv4, it would require one or two bytes more than the COMPRESSED_UDP form listed above in order to carry the IPv4 ID field. For IPv6, there is no ID field and this non-differential compression is more resilient to packet loss.

Assignments of numeric codes for these packet formats in the Point-to-Point Protocol [4] are to be made by the Internet Assigned Numbers Authority.

3.3.1. FULL_HEADER (uncompressed) packet format

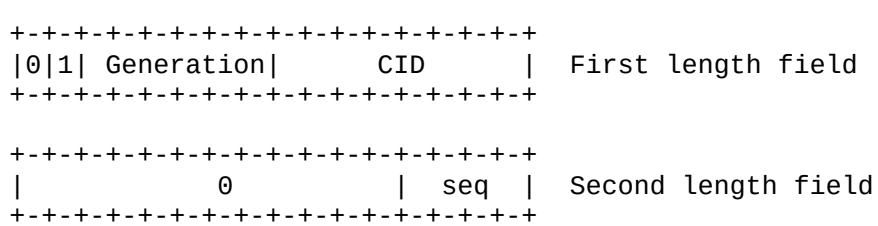
The definition of the FULL_HEADER packet given here is intended to be the consistent with the definition given in [3]. Full details on design choices are given there.

The format of the FULL_HEADER packet is the same as that of the original packet. In the IPv4 case, this is usually an IP header, followed by a UDP header and UDP payload that may be an RTP header and its payload. However, the FULL_HEADER packet may also carry IP encapsulated packets, in which case there would be two IP headers followed by UDP and possibly RTP. Or in the case of IPv6, the packet may be built of some combination of IPv6 and IPv4 headers. Each successive header is indicated by the type field of the previous header, as usual.

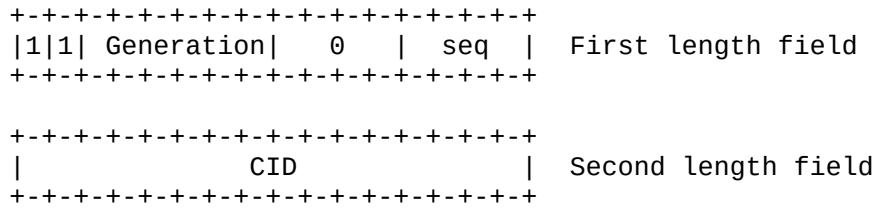
The FULL_HEADER packet differs from the corresponding normal IPv4 or IPv6 packet in that it must also carry the compression context ID and the 4-bit sequence number. In order to avoid expanding the size of the header, these values are inserted into length fields in the IP and UDP headers since the actual length may be inferred from the length provided by the link layer. Two 16-bit length fields are needed; these are taken from the first two available headers in the packet. That is, for an IPv4/UDP packet, the first length field is the total length field of the IPv4 header, and the second is the length field of the UDP header. For an IPv4 encapsulated packet, the first length field would come from the total length field of the first IP header, and the second length field would come from the total length field of the second IP header.

As specified in Sections 5.3.2 of [3], the position of the context ID (CID) and 4-bit sequence number varies depending upon whether 8- or 16-bit context IDs have been selected, as shown in the following diagram (16 bits wide, with the most-significant bit is to the left):

For 8-bit context ID:



For 16-bit context ID:



The first bit in the first length field indicates the length of the CID. The length of the CID MUST either be constant for all contexts or two additional distinct packet types MUST be provided to separately indicate COMPRESSED_UDP and COMPRESSED_RTP packet formats with 8- and 16-bit CIDs. The second bit in the first length field is 1 to indicate that the 4-bit sequence number is present, as is always the case for this IP/UDP/RTP compression scheme.

The generation field is used with IPv6 for COMPRESSED_NON_TCP packets as described in [3]. For IPv4-only implementations that do not use COMPRESSED_NON_TCP packets, the compressor SHOULD set the generation value to zero. For consistent operation between IPv4 and IPv6, the generation value is stored in the context when it is received by the decompressor, and the most recent value is returned in the CONTEXT_STATE packet.

When a FULL_HEADER packet is received, the complete set of headers is stored into the context selected by the context ID. The 4-bit sequence number is also stored in the context, thereby resynchronizing the decompressor to the compressor.

When COMPRESSED_NON_TCP packets are used, the 4-bit sequence number is inserted into the "Data Field" of that packet and the D bit is set as described in Section 6 of [3]. When a COMPRESSED_NON_TCP packet is received, the generation number is compared to the value stored in the context. If they are not the same, the context is not up to date and MUST be refreshed by a FULL_HEADER packet. If the generation does match, then the compressed IP and UDP header information, the 4-bit sequence number, and the (potential) RTP header are all stored into the saved context.

The amount of memory required to store the context will vary depending upon how many encapsulating headers are included in the FULL_HEADER packet. The compressor and decompressor MAY negotiate a maximum header size.

3.3.2. COMPRESSED_RTP packet format

When the second-order difference of the RTP header from packet to packet is zero, the decompressor can reconstruct a packet simply by adding the stored first-order differences to the stored uncompressed header representing the previous packet. All that need be communicated is the session context identifier and a small sequence number (not related to the RTP sequence number) to maintain synchronization and detect packet loss between the compressor and decompressor.

If the second-order difference of the RTP header is not zero for some fields, the new first-order difference for just those fields is communicated using a compact encoding. The new first-order difference values are added to the corresponding fields in the uncompressed header in the decompressor's session context, and are also stored explicitly in the context to be added to the corresponding fields again on each subsequent packet in which the second-order difference is zero. Each time the first-order difference changes, it is transmitted and stored in the context.

In practice, the only fields for which it is useful to store the first-order difference are the IPv4 ID field and the RTP timestamp. For the RTP sequence number field, the usual increment is 1. If the sequence number changes by other than 1, the difference must be communicated but does not set the expected difference for the next packet. Instead, the expected first-order difference remains fixed at 1 so that the difference need not be explicitly communicated on the next packet assuming it is in order.

For the RTP timestamp, when a FULL_HEADER, COMPRESSED_NON_TCP or COMPRESSED_UDP packet is sent to refresh the RTP state, the stored first-order difference is initialized to zero. If the timestamp is the same on the next packet (e.g., same video frame), then the second-order difference is zero. Otherwise, the difference between the timestamps of the two packets is transmitted as the new first-order difference to be added to the timestamp in the uncompressed header stored in the decompressor's context and also stored as the first-order difference in that context. Each time the first-order difference changes on subsequent packets, that difference is again transmitted and used to update the context.

Similarly, since the IPv4 ID field frequently increments by one, the first-order difference for that field is initialized to one when the state is refreshed by a FULL_HEADER packet, or when a COMPRESSED_NON_TCP packet is sent since it carries the ID field in uncompressed form. Thereafter, whenever the first-order difference changes, it is transmitted and stored in the context.

A bit mask will be used to indicate which fields have changed by other than the expected difference. In addition to the small link sequence number, the list of items to be conditionally communicated in the compressed IP/UDP/RTP header is as follows:

I = IPv4 packet ID (always 0 if no IPv4 header)
U = UDP checksum
M = RTP marker bit
S = RTP sequence number
T = RTP timestamp
L = RTP CSRC count and list

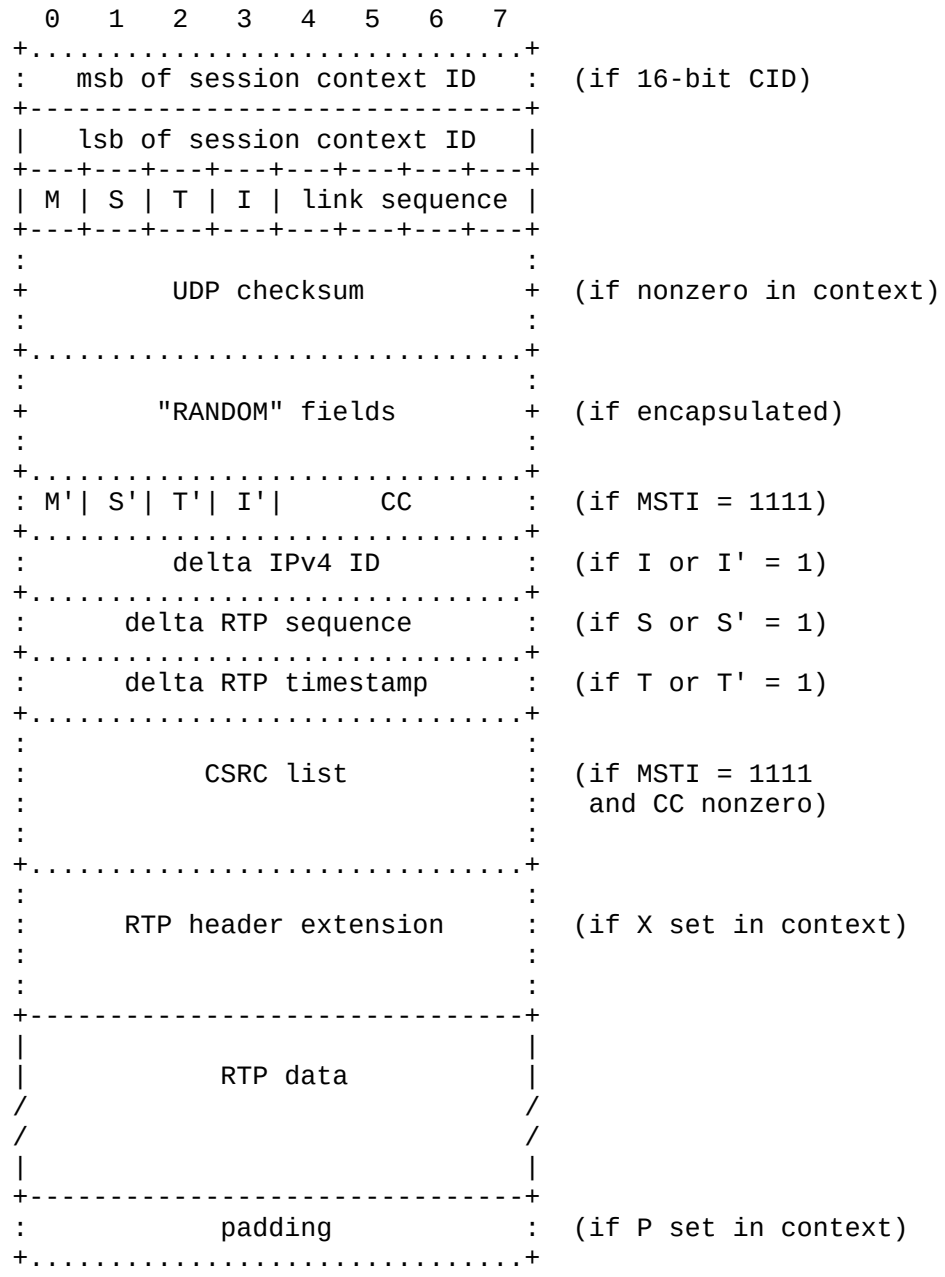
If 4 bits are needed for the link sequence number to get a reasonable probability of loss detection, there are too few bits remaining to assign one bit to each of these items and still fit them all into a single byte to go along with the context ID.

It is not necessary to explicitly carry the U bit to indicate the presence of the UDP checksum because a source will typically include check-sums on all packets of a session or none of them. When the session state is initialized with an uncompressed header, if there is a nonzero checksum present, an unencoded 16-bit checksum will be inserted into the compressed header in all subsequent packets until this setting is changed by sending another uncompressed packet.

Of the remaining items, the L bit for the CSRC count and list may be the one least frequently used. Rather than dedicating a bit in the mask to indicate CSRC change, an unusual combination of the other bits may be used instead. This bit combination is denoted MSTI. If all four of the bits for the IP packet ID, RTP marker bit, RTP sequence number and RTP timestamp are set, this is a special case indicating an extended form of the compressed RTP header will follow. That header will include an additional byte containing the real values of the four bits plus the CC count. The CSRC list, of length indicated by the CC count, will be included just as it appears in the uncompressed RTP header.

The other fields of the RTP header (version, P bit, X bit, payload type and SSRC identifier) are assumed to remain relatively constant. In particular, the SSRC identifier is defined to be constant for a given context because it is one of the factors selecting the context. If any of the other fields change, the uncompressed RTP header MUST sent as described in Section 3.3.3.

The following diagram shows the compressed IP/UDP/RTP header with dotted lines indicating fields that are conditionally present. The most significant bit is numbered 0. Multi-byte fields are sent in network byte order (most significant byte first). The delta fields are often a single byte as shown but may be two or three bytes depending upon the delta value as explained in Section 3.3.4.

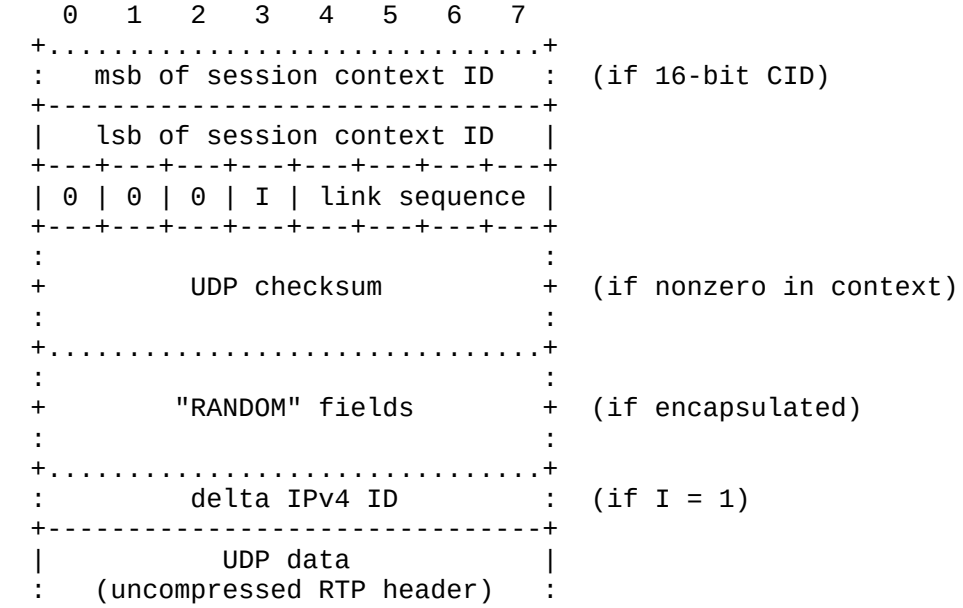


When more than one IPv4 header is present in the context as initialized by the FULL_HEADER packet, then the IP ID fields of encapsulating headers MUST be sent as absolute values as described in

[3]. These fields are identified as "RANDOM" fields. They are inserted into the COMPRESSED_RTP packet in the same order as they appear in the original headers, immediately following the UDP checksum if present or the MSTI byte if not, as shown in the diagram. Only if an IPv4 packet immediately precedes the UDP header will the IP ID of that header be sent differentially, i.e., potentially with no bits if the second difference is zero, or as a delta IPv4 ID field if not. If there is not an IPv4 header immediately preceding the UDP header, then the I bit MUST be 0 and no delta IPv4 ID field will be present.

3.3.3. COMPRESSED_UDP packet format

If there is a change in any of the fields of the RTP header that are normally constant (such as the payload type field), then an uncompressed RTP header MUST be sent. If the IP and UDP headers do not also require updating, this RTP header MAY be carried in a COMPRESSED_UDP packet rather than a FULL_HEADER packet. The COMPRESSED_UDP packet has the same format as the COMPRESSED_RTP packet except that the M, S and T bits are always 0 and the corresponding delta fields are never included:



Note that this constitutes a form of IP/UDP header compression different from COMPRESSED_NON_TCP packet type defined in [3]. The motivation is to allow reaching the target of two bytes when UDP checksums are disabled, as IPv4 allows. The protocol in [3] does not use differential coding for UDP packets, so in the IPv4 case, two

bytes of IP ID, and two bytes of UDP checksum if nonzero, would always be transmitted in addition to two bytes of compression prefix. For IPv6, the COMPRESSED_NON_TCP packet type MAY be used instead.

3.3.4. Encoding of differences

The delta fields in the COMPRESSED_RTP and COMPRESSED_UDP packets are encoded with a variable-length mapping for compactness of the more commonly-used values. A default encoding is specified below, but it is RECOMMENDED that implementations use a table-driven delta encoder and decoder to allow negotiation of a table specific for each session if appropriate, possibly even an optimal Huffman encoding. Encodings based on sequential interpretation of the bit stream, of which this default table and Huffman encoding are examples, allow a reasonable table size and may result in an execution speed faster than a non-table-driven implementation with explicit tests for ranges of values.

The default delta encoding is specified in the following table. This encoding was designed to efficiently encode the small changes that may occur in the IP ID and in RTP sequence number when packets are lost upstream from the compressor, yet still handling most audio and video deltas in two bytes. The column on the left is the decimal value to be encoded, and the column on the right is the resulting sequence of bytes shown in hexadecimal and in the order in which they are transmitted (network byte order). The first and last values in each contiguous range are shown, with ellipses in between:

Decimal	Hex
-16384	C0 00 00
:	:
-129	C0 3F 7F
-128	80 00
:	:
-1	80 7F
0	00
:	:
127	7F
128	80 80
:	:
16383	BF FF
16384	C0 40 00
:	:
4194303	FF FF FF

For positive values, a change of zero through 127 is represented directly in one byte. If the most significant two bits of the byte are 10 or 11, this signals an extension to a two- or three-byte

value, respectively. The least significant six bits of the first byte are combined, in decreasing order of significance, with the next one or two bytes to form a 14- or 22-bit value.

Negative deltas may occur when packets are misordered or in the intentionally out-of-order RTP timestamps on MPEG video [5]. These events are less likely, so a smaller range of negative values is encoded using otherwise redundant portions of the positive part of the table.

A change in the RTP timestamp value less than -16384 or greater than 4194303 forces the RTP header to be sent uncompressed using a FULL_HEADER, COMPRESSED_NON_TCP or COMPRESSED_UDP packet type. The IP ID and RTP sequence number fields are only 16 bits, so negative deltas for those fields SHOULD be masked to 16 bits and then encoded (as large positive 16-bit numbers).

3.3.5. Error Recovery

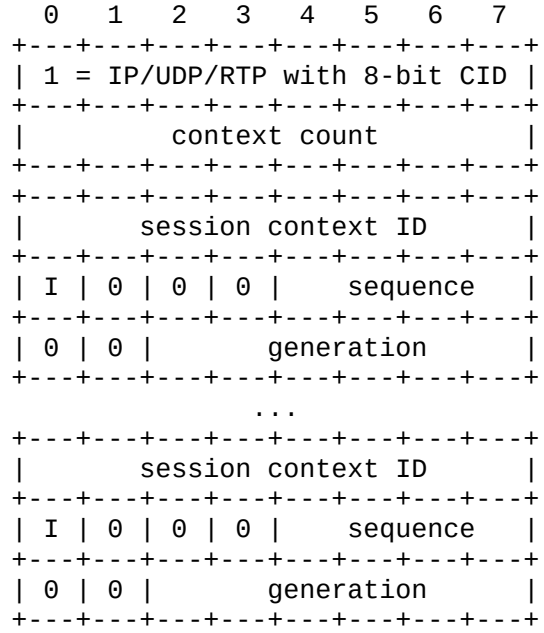
Whenever the 4-bit sequence number for a particular context increments by other than 1, except when set by a FULL_HEADER or COMPRESSED_NON_TCP packet, the decompressor MUST invalidate that context and send a CONTEXT_STATE packet back to the compressor indicating that the context has been invalidated. All packets for the invalid context MUST be discarded until a FULL_HEADER or COMPRESSED_NON_TCP packet is received for that context to re-establish consistent state (unless the "twice" algorithm is used as described later in this section). Since multiple compressed packets may arrive in the interim, the decompressor SHOULD NOT retransmit the CONTEXT_STATE packet for every compressed packet received, but instead SHOULD limit the rate of retransmission to avoid flooding the reverse channel.

When an error occurs on the link, the link layer will usually discard the packet that was damaged (if any), but may provide an indication of the error. Some time may elapse before another packet is delivered for the same context, and then that packet would have to be discarded by the decompressor when it is observed to be out of sequence, resulting in at least two packets lost. To allow faster recovery if the link does provide an explicit error indication, the decompressor MAY optionally send an advisory CONTEXT_STATE packet listing the last valid sequence number and generation number for one or more recently active contexts (not necessarily all). For a given context, if the compressor has sent no compressed packet with a higher sequence number, and if the generation number matches the current generation, no corrective action is required. Otherwise, the compressor MAY choose to mark the context invalid so that the next packet is sent in FULL_HEADER or COMPRESSED_NON_TCP mode (FULL_HEADER

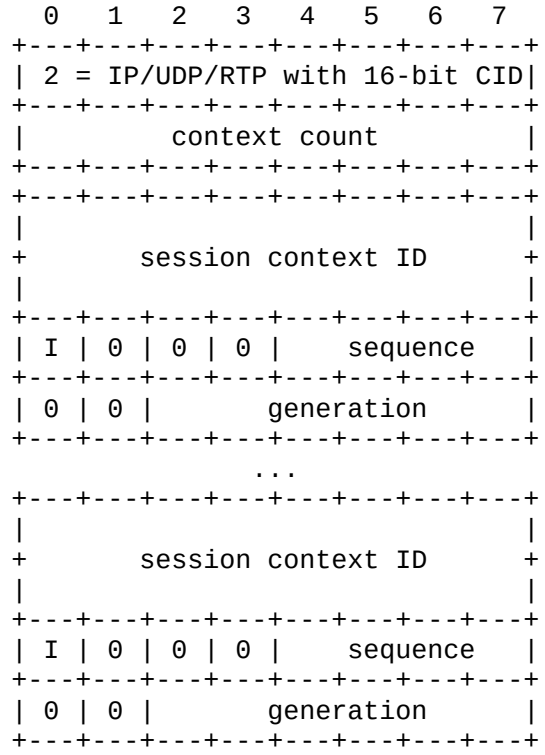
is required if the generation doesn't match). However, note that if the link round-trip-time is large compared to the inter-packet spacing, there may be several packets from multiple contexts in flight across the link, increasing the probability that the sequence numbers will already have advanced when the CONTEXT_STATE packet is received by the compressor. The result could be that some contexts are invalidated unnecessarily, causing extra bandwidth to be consumed.

The format of the CONTEXT_STATE packet is shown in the following diagrams. The first byte is a type code to allow the CONTEXT_STATE packet type to be shared by multiple compression schemes within the general compression framework specified in [3]. The contents of the remainder of the packet depends upon the compression scheme. For the IP/UDP/RTP compression scheme specified here, the remainder of the CONTEXT_STATE packet is structured as a list of blocks to allow the state for multiple contexts to be indicated, preceded by a one-byte count of the number of blocks.

Two type code values are used for the IP/UDP/RTP compression scheme. The value 1 indicates that 8-bit session context IDs are being used:



The value 2 indicates that 16-bit session context IDs are being used. The session context ID is sent in network byte order (most significant byte first):



The bit labeled "I" is set to one for contexts that have been marked invalid and require a FULL_HEADER of COMPRESSED_NON_TCP packet to be transmitted. If the I bit is zero, the context state is advisory. The I bit is set to zero to indicate advisory context state that MAY be sent following a link error indication.

Since the CONTEXT_STATE packet itself may be lost, retransmission of one or more blocks is allowed. It is expected that retransmission will be triggered only by receipt of another packet, but if the line is near idle, retransmission MAY be triggered by a relatively long timer (on the order of 1 second).

If a CONTEXT_STATE block for a given context is retransmitted, it may cross paths with the FULL_HEADER or COMPRESSED_NON_TCP packet intended to refresh that context. In that case, the compressor MAY choose to ignore the error indication.

In the case where UDP checksums are being transmitted, the decompressor MAY attempt to use the "twice" algorithm described in section 10.1 of [3]. In this algorithm, the delta is applied more than once on the assumption that the delta may have been the same on the missing packet(s) and the one subsequently received. Success is

indicated by a checksum match. For the scheme defined here, the difference in the 4- bit sequence number tells number of times the delta must be applied. Note, however, that there is a nontrivial risk of an incorrect positive indication. It may be advisable to request a FULL_HEADER or COMPRESSED_NON_TCP packet even if the "twice" algorithm succeeds.

Some errors may not be detected, for example if 16 packets are lost in a row and the link level does not provide an error indication. In that case, the decompressor will generate packets that are not valid. If UDP checksums are being transmitted, the receiver will probably detect the invalid packets and discard them, but the receiver does not have any means to signal the decompressor. Therefore, it is RECOMMENDED that the decompressor verify the UDP checksum periodically, perhaps one out of 16 packets. If an error is detected, the decompressor would invalidate the context and signal the compressor with a CONTEXT_STATE packet.

3.4. Compression of RTCP Control Packets

By relying on the RTP convention that data is carried on an even port number and the corresponding RTCP packets are carried on the next higher (odd) port number, one could tailor separate compression schemes to be applied to RTP and RTCP packets. For RTCP, the compression could apply not only to the header but also the "data", that is, the contents of the different packet types. The numbers in Sender Report (SR) and Receiver Report (RR) RTCP packets would not compress well, but the text information in the Source Description (SDES) packets could be compressed down to a bit mask indicating each item that was present but compressed out (for timing purposes on the SDES NOTE item and to allow the end system to measure the average RTCP packet size for the interval calculation).

However, in the compression scheme defined here, no compression will be done on the RTCP headers and "data" for several reasons (though compression SHOULD still be applied to the IP and UDP headers). Since the RTP protocol specification suggests that the RTCP packet interval be scaled so that the aggregate RTCP bandwidth used by all participants in a session will be no more than 5% of the session bandwidth, there is not much to be gained from RTCP compression. Compressing out the SDES items would require a significant increase in the shared state that must be stored for each context ID. And, in order to allow compression when SDES information for several sources was sent through an RTP "mixer", it would be necessary to maintain a separate RTCP session context for each SSRC identifier. In a session with more than 255 participants, this would cause perfect thrashing of the context cache even when only one participant was sending data.

Even though RTCP is not compressed, the fraction of the total bandwidth occupied by RTCP packets on the compressed link remains no more than 5% in most cases, assuming that the RTCP packets are sent as COMPRESSED_UDP packets. Given that the uncompressed RTCP traffic consumes no more than 5% of the total session bandwidth, then for a typical RTCP packet length of 90 bytes, the portion of the compressed bandwidth used by RTCP will be no more than 5% if the size of the payload in RTP data packets is at least 108 bytes. If the size of the RTP data payload is smaller, the fraction will increase, but is still less than 7% for a payload size of 37 bytes. For large data payloads, the compressed RTCP fraction is less than the uncompressed RTCP fraction (for example, 4% at 1000 bytes).

3.5. Compression of non-RTP UDP Packets

As described earlier, the COMPRESSED_UDP packet MAY be used to compress UDP packets that don't carry RTP. Whatever data follows the UDP header is unlikely to have some constant values in the bits that correspond to usually constant fields in the RTP header. In particular, the SSRC field would likely change. Therefore, it is necessary to keep track of the non-RTP UDP packet streams to avoid using up all the context slots as the "SSRC field" changes (since that field is part of what identifies a particular RTP context). Those streams may each be given a context, but the encoder would set a flag in the context to indicate that the changing SSRC field should be ignored and COMPRESSED_UDP packets should always be sent instead of COMPRESSED_RTP packets.

4. Interaction With Segmentation

A segmentation scheme may be used in conjunction with RTP header compression to allow small, real-time packets to interrupt large, presumably non-real-time packets in order to reduce delay. It is assumed that the large packets bypass the compressor and decompressor since the interleaving would modify the sequencing of packets at the decompressor and cause the appearance of errors. Header compression should be less important for large packets since the overhead ratio is smaller.

If some packets from an RTP session context are selected for segmentation (perhaps based on size) and some are not, there is a possibility of re-ordering. This would reduce the compression efficiency because the large packets would appear as lost packets in the sequence space. However, this should not cause more serious problems because the RTP sequence numbers should be reconstructed correctly and will allow the application to correct the ordering.

Link errors detected by the segmentation scheme using its own sequencing information MAY be indicated to the compressor with an advisory CONTEXT_STATE message just as for link errors detected by the link layer itself.

The context ID byte is placed first in the COMPRESSED_RTP header so that this byte MAY be shared with the segmentation layer if such sharing is feasible and has been negotiated. Since the compressor may assign context ID values arbitrarily, the value can be set to match the context identifier from the segmentation layer.

5. Negotiating Compression

The use of IP/UDP/RTP compression over a particular link is a function of the link-layer protocol. It is expected that such negotiation will be defined separately for PPP [4], for example. The following items MAY be negotiated:

- The size of the context ID.
- The maximum size of the stack of headers in the context.
- A context-specific table for decoding of delta values.

6. Acknowledgments

Several people have contributed to the design of this compression scheme and related problems. Scott Petrack initiated discussion of RTP header compression in the AVT working group at Los Angeles in March, 1996. Carsten Bormann has developed an overall architecture for compression in combination with traffic control across a low-speed link, and made several specific contributions to the scheme described here. David Oran independently developed a note based on similar ideas, and suggested the use of PPP Multilink protocol for segmentation. Mikael Degermark has contributed advice on integration of this compression scheme with the IPv6 compression framework.

7. References:

[1] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for real-time applications", RFC 1889, January 1996.

[2] Jacobson, V., "TCP/IP Compression for Low-Speed Serial Links", RFC 1144, February 1990.

[3] Degermark, M., Nordgren, B. and S. Pink, "Header Compression for IPv6", RFC 2507, February 1999.

[4] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[5] Hoffman, D., Fernando, G., Goyal, V. and M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video", RFC 2250, January 1998.

8. 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. 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 [3].

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.

A security review of this protocol found no additional security considerations.

9. Authors' Addresses

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

EMail: casner@cisco.com

Van Jacobson
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
United States

EMail: van@cisco.com

10. Full Copyright Statement