Robust Header Compression
Internet-Draft
Intended status: Standards Track
Expires: August 30, 2007

C. Bormann
Universitaet Bremen TZI

February 26, 2007

A ROHC Profile for RTCP (ROHC-RTCP)
draft-bormann-rohc-avt-rtcp-profile-00

Status of this Memo

Copyright Notice

Abstract

This document specifies a ROHC (Robust Header Compression) profile for compression of RTCP packets. The profile, called ROHC-RTCP, provides efficient and robust compression of RTCP packets, including frequently used RTCP features.

ROHC-RTCP is intended to be used in conjunction with ROHC profiles for the compression of RTP headers such as RFC 3905 and ROHCv2, making use of context replication (RFC 4164) where appropriate. In

particular, ROHC-RTCP is intended to remove the need for standardization of special-purpose variants of RTCP for applications just because these also need to work over low-speed links.

$Id: draft-bormann-rohc-avt-rtcp-profile.xml,v 1.11 2007/02/26 13:36:24 cabo Exp $

Table of Contents

1. Introduction

When the RTP profiles were defined for the ROHC framework, it was considered unnecessary to provide compression for RTP's often neglected child, RTCP. RTCP is designed to use only a small fraction of the session bandwidth; any improvements from compressing it would only occur within this small fraction. In addition, properly compressing RTCP raised many questions that the ROHC designers were not ready to answer in 2000.

Meanwhile, many creative uses for RTCP have been invented [RFC4585][I-D.ietf-avt-rtcpxr-video]. Also, with the development of context replication [RFC4164], the separation of the ROHC framework [I-D.ietf-rohc-rfc3095bis-framework], and the development of ROHCv2 profiles [I-D.ietf-rohc-rfc3095bis-rohcv2-profiles], the time may be ripe to address RTCP header compression.

This document defines a ROHC profile for compression of RTCP packets. Note that there is little that could be called an RTCP header; instead, for the purposes of this document, the entire RTCP packet will be considered header information.

The profile indeed is not specific to RTCP: It could theoretically be used to compress other recurring control information transmitted via UDP. However, where design decisions have been made based on the characteristics of this control information, this specification focuses on RTCP.

2. Terminology

- State Item

  Sequence of bytes that is established as a common state between compressor and decompressor using ROHC methods. Within an RTCP compression context, there may be multiple state items, which are indexed by a state identifier.

- State Identifier

  8-bit identifier for one of the state items that is part of the current context.

3. Protocol Operation

This section gives an overview of the operation of ROHC-RTCP.

The ROHC-RTCP protocol behaves exactly like the ROHCv2 UDP protocol, with the exceptions listed below.

3.1. Creating Contexts

A ROHC-RTCP context is created using an IR (initialization/refresh) or a CR (context replication) packet (the latter is particularly useful where an RTP HC context already exists for the RTP session).

For IR packets, the context created contains exactly the information a ROHCv2 UDP context would contain, with the addition of a delta format (see below). For CR packets, the provisions of [RFC4164] are an integral part of this specification; again, the replicated context is enhanced by a delta format.

When the context is created, a delta format is established for this context; to minimize the change to the ROHCv2 headers, this is specified in the Initial Byte of the Body Format, which replaces the payload in all packet formats derived from ROHCv2.

3.2. Creating State

Every ROHC-RTCP packet supplies an 8-bit target state identifier. The result of decompressing the packet is kept for future reference under the state identifier supplied; any previous state item under this state identifier is replaced.

Normal ROHC considerations apply about when a compressor can start to rely on a state item to be present at the decompressor. As it is based on ROHCv2, ROHC-RTCP supports the equivalent of unidirectional and optimistic establishment of state items.

3.3. Using State

CO-type ROHC-RTCP packets reference an 8-bit state identifier and supply a delta to the state item referenced as well as an 8-bit CRC. The packet is decompressed by applying the supplied delta to the state item referenced. If no state item exists at the decompressor under the state identifier given or if checking the 8-bit CRC against the decompressed packet fails, negative feedback is sent back to the compressor (as described in the base ROHCv2 specification) and the decompressed packet is discarded.

4. Packet Formats

This section defines the packet formats used in ROHC-RTCP.

4.1. Base Packet Formats

The ROHC-RTCP base packet format is exactly like the ROHCv2 UDP format [I-D.ietf-rohc-rfc3095bis-rohcv2-profiles] with the following exceptions:

1. There is a CR format, defined below.
2. Instead of the payload as defined in ROHCv2-UDP, the ROHC-RTCP body format, defined below, is carried in ROHC-RTCP packets.

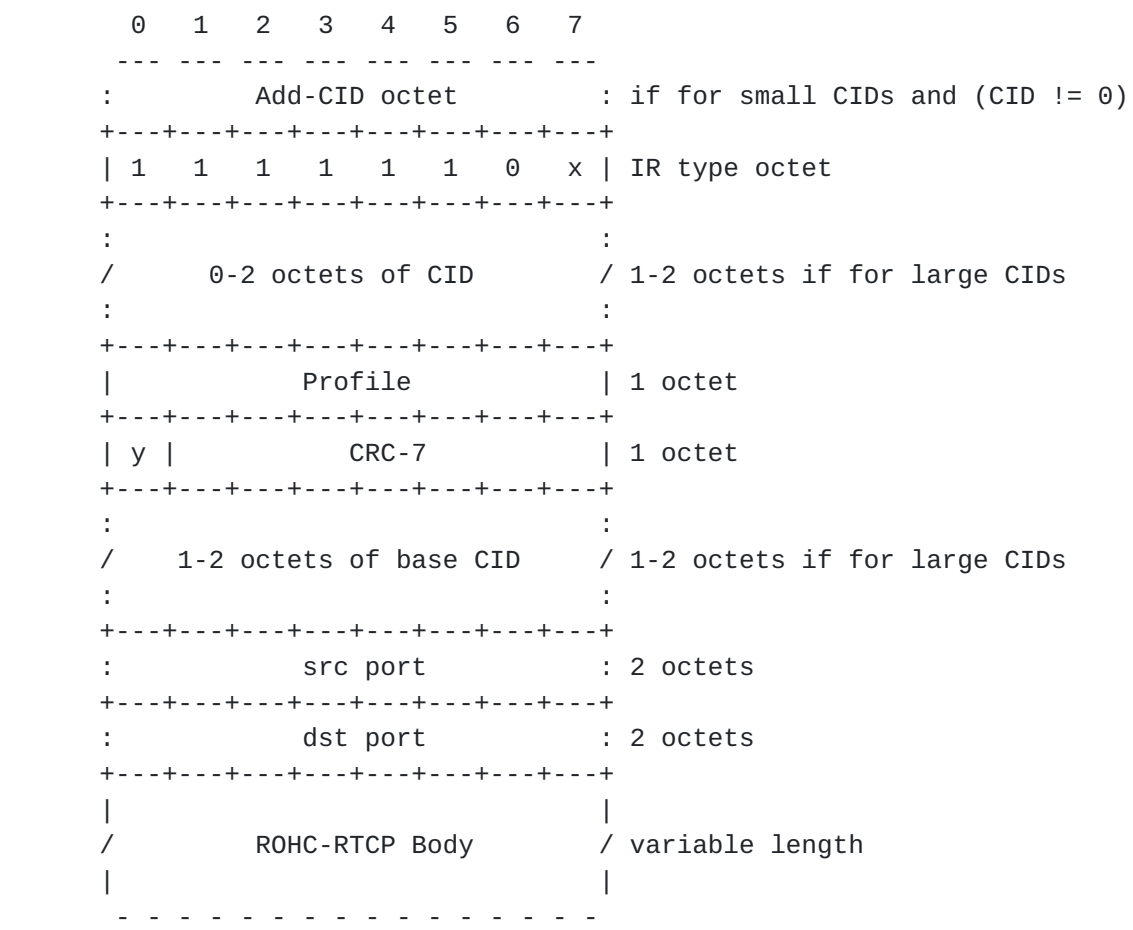
4.2. CR Format

A ROHC-RTCP CR header that references a ROHCv2 UDP context replicates it as the basis for the ROHC-RTCP context.

A ROHC-RTCP CR header that references a [RFC3095] UDP context replicates it as the basis for the ROHC-RTCP context, converting the [RFC3095] UDP state to a ROHCv2 UDP state by discarding mode information and converting the SN to an MSN.

A ROHC-RTCP CR header that references a ROHCv2 RTP context or a [RFC3095] RTP context replicates/converts it as the basis for the ROHC-RTCP context, removing the RTP specific information (and creating an MSN out of the RTP SN for RFC3095).

The UDP part of the static chain of the context is then modified by the port number pair given in the ROHC-RTCP CR header; all other fields of the IP and the UDP headers are replicated from the base context.



For this version of the specification, x == 0 and y == 0.

4.3. ROHC-RTCP Body Format

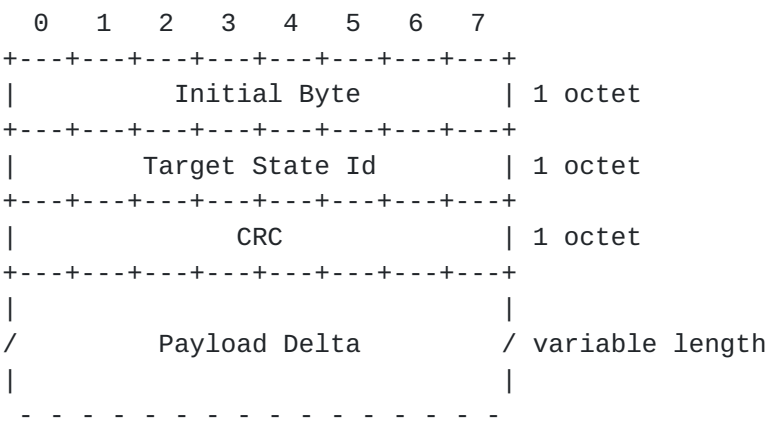
ROHC-RTCP packets contain a ROHC-RTCP Body.

For CO packets, the ROHC-RTCP Body references a state item, the identifier of which is given by the Initial Byte. For IR and CR packets, the referenced state item is empty (a string of zero bytes).

(TBD: For CR, switching from ROHC-UDP to ROHC-RTCP may actually make sense; in this case it may be useful to construct a different state item, but this would assume there is information about a payload available in a ROHC-UDP context.)

For IR and CR packets, the Initial Byte instead specifies the Delta Format that is to be used for this context.

For the decompressed packet, the payload is reconstructed from the state item and the payload delta using the delta format given (for IR and CR packets) or established for this context (for CO packets).



4.4. Delta Format

This specification only defines Delta Format 0x00. Additional Delta Formats can be defined later; see Section 7 below.

Delta Format 0x00 is a sequence of byte codes, each of which contributes to the output and/or modifies the interpretation state in one of the following ways::

1. Supplies additional bits to operands of subsequent byte codes.
2. Copies a specified number of bytes from the current state item to the output and moves the current position in the current state item forward by this number.
3. Copies a specified number of bytes from the Body itself to the output, which are then skipped during interpretation ("arguments"). The specified number of bytes must follow the bytecode in the Body; an underrun causes decompression failure.
4. Modifies a specified number (m) of bytes from the state item in a specified way: the m bytes are interpreted as a MSB-first binary number and a number n is added to that number modulo 256^m; the resulting number is copied in MSB-first form to the output.
5. Selects a different state item as the current state item.
6. Modifies the current position in the current state item in a specified way, e.g., skips a specified number of bytes in the state item (this position is only relevant for the purposes of this single instance of delta format interpretation -- the referenced state item is not itself modified).

The interpretation of bytecodes is finished (and the resulting output processed as a payload) when the end of the Body is reached during interpretation.

During interpretation of the byte codes, the decompressor maintains a current position in each referenced state item.

If a byte code cannot be meaningfully interpreted, e.g. when the

bytes in the state item or the body run out prematurely, this causes decompression failure, i.e. is treated like a CRC failure.

In the following table, some operations make use of a number n. The number n is computed by appending the nnn bits in the current bytecode to any accumulated prefix and adding one to the result. The accumulated prefix starts out as an empty bitstring and is appended to by one specific bytecode. If the concatenation of the accumulated prefix and the nnn bits exceeds 16 bits, this is treated as decompression failure. If there is an s bit in the bytecode, n is negated (n = -n) if s == 1. Computing n always clears the accumulated prefix value, if any. All bit operations are performed most significant bit first.

Some operations make use of a number m. This number is computed from mm as follows: m = mm+1.

Operations that use the "CSI" use the current state item, at the current position noted for the current state item. The interpreter maintains a separate position for the each state item that is used in the bytecode; all positions are initialized to zero at the start of interpreting each Body. Operations on the CSI number are modulo 256 (i.e., the CSI number is an 8-bit number). Operations on the position in the CSI are modulo the size of the CSI in bytes.

Bytecode	Arguments	Semantics
000nnnnn	(none)	Append nnnnn to the accumulated prefix.
001nnnnn	(none)	Copy n bytes from CSI to output.
010nnnnn	n bytes	Copy n bytes from Body to output.
011mmsnn	(none)	Take m bytes from CSI, add n, output as m bytes.
10snnnnn	(none)	Add n to the CSI number, use as new CSI number.
11snnnnn	(none)	Change position in CSI by n, modulo size(CSI).

Note that implementations can represent n (including the accumulated prefix) as a 16-bit value, as operations that would need more bits are not meaningful or would even cause decompression failure. Interpreting the change to the CSI position modulo the size of the CSI (in bytes) means that all possible bit patterns are meaningful (e.g., going beyond the end starts at the beginning again); to simplify implementation of this case, accumulating large (more than 16-bit) values of n is simply disallowed (causes decompression failure).

5. TBD

The following issues should be resolved for -01:

1. Delta formats are not yet subject to negotiation.
2. Is it entirely clear from the text that state items are per context? Is that the right decision?
3. There probably needs to be some text about static/dynamic chains in the packet formats.
4. Delta Format 0: It should be possible to refer to the packet under construction as if it were a state item. (Another copy-n instruction? And a goto instruction? Or clearing out the target state ID? State ID 0?)
5. Do we really need two CRCs for CR?
6. Appendix A needs to be written.

6. Security considerations

The security considerations of [RFC3095] apply. An additional denial-of-service attack is possible in ROHC-RTCP by filling up the receiver with state items. The decompressor must be careful about range-checking the Delta Format bytecode operations. As with all compression protocols, the expansion that is likely to occur in decompression can be used to amplify a denial-of-service attack. (Note that bytecode interpretation is linear or sublinear to the length of the bytecode -- there is no attack vector of overloading the decompressor's CPU by sending looping or otherwise malformed bytecode.)

7. IANA Considerations

The ROHC profile identifier 0x00XX [# Editor's Note: To be replaced before publication #] has been reserved by the IANA for the profile defined in this document.

The ROHC-RTCP delta format identifier 0x00 has been reserved by the IANA for the ROHC-RTCP delta format 0x00 defined in this document.

[# Editor's Note: rest of this section to be removed before publication: #]

A ROHC profile identifier must be reserved by the IANA for the new profile defined in this document. A suggested registration in the "RObust Header Compression (ROHC) Profile Identifiers" name space would then be:

Profile	Usage	Reference
0x0009	ROHC RTCP	[RFC XXXX (this)]

Author's note: This suggestion must be updated before sending to

IANA.

A new IANA registry "ROHC-RTCP delta formats" is established by this specification under the regime "Standards Action required". Initially, the value 0x00 is defined by this specification; further values to be defined later need to be 8-bit numbers (values between 0x01 and 0xFF) and the pertinent standard needs to define the semantics of the delta format to a similar effect as in Section 4.4 of this specification.

8. Contributors

The author would like to thank Joerg Ott, who noticed that this profile was finally needed and supplied some sample data to be used in appendix A.

Ghyslain Pelletier supplied comments that significantly shaped this specification. Kristofer Sandlund's comments also pointed me in the right direction.

9. Acknowledgements

A number of important concepts and ideas have been borrowed from ROHC [RFC3095] and the ROHCv2 Internet-Drafts.

10. References

10.1. Normative References

[I-D.ietf-rohc-rfc3095bis-framework]
Jonsson, L., "The RObust Header Compression (ROHC) Framework", draft-ietf-rohc-rfc3095bis-framework-01 (work in progress), July 2006.

[I-D.ietf-rohc-rfc3095bis-rohcv2-profiles]
Pelletier, G. and K. Sandlund, "RObust Header Compression Version 2 (RoHCv2): Profiles for RTP, UDP, IP, ESP and UDP Lite", draft-ietf-rohc-rfc3095bis-rohcv2-profiles-00 (work in progress), September 2006.

[RFC4164] Pelletier, G., "RObust Header Compression (ROHC): Context Replication for ROHC Profiles", RFC 4164, August 2005.

10.2. Informative References

[I-D.ietf-avt-rtcpxr-video]
Clark, A. and A. Pendleton, "RTCP XR - IP Video Metrics Report Blocks", draft-ietf-avt-rtcpxr-video-00 (work in progress), December 2006.

[RFC3095] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., 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.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

Appendix A. Bit-level Worked Example

This appendix gives a worked example at the bit level, showing how various forms of RTCP packets are compressed by ROHC-RTCP.

TBD

Appendix B. Reference Implementation

This appendix gives a reference implementation of the delta format decompressor. This appendix is intended to elucidate the normative definition in Section 4.4

```
#!/usr/bin/env ruby
#
# $Id: decomp.rb,v 1.10 2007/02/25 13:27:48 cabo Exp $
#
class DecompressionFailure < StandardError
end

class ROHC_RTCP
  def initialize(delta_format = 0)
    @delta_format = delta_format
    @state_items = Hash.new { |h, k| h[k] = '' }
  end
  def state_item_is(n, si)
    raise "Bad state item number #{n}" unless n >= 0 && n < 256
```

```
      @state_items[n] = si
    end
    def pos
      @state_pos[@csinum]
    end
    def pos=(np)
      @state_pos[@csinum] = np
    end
    def csi
      @state_items[@csinum]
    end
    def read_csi(nb)
      out = csi[pos, nb]
      self.pos += nb
      raise DecompressionFailure unless out.size == nb
      out
    end
    def n(op, bits, s = 0)
      out = (@accum << bits) | (op & ((1 << bits) - 1))
      @accum = 0
      raise DecompressionFailure unless out < 0x10000 # 16 bits
      out += 1
      out = -out if s[bits] != 0
      out
    end
    def decompress(bo, refsi)
      raise DecompressionFailure unless @delta_format == 0
      out = ''
      @accum = 0
      @csinum = refsi
      @state_pos = Hash.new(0)
      i = 0
      while i < bo.size
        op = bo[i]; i += 1
        case op >> 5
        when 0b000
          @accum = (@accum << 5) | (op & ((1 << 5) - 1))
        when 0b001
          out << read_csi(n(op, 5))
        when 0b010
          nb = n(op, 5)
          out << bo[i, nb]
          i += nb
          raise DecompressionFailure unless i <= bo.size
        when 0b011
          delta = n(op, 2, op)
          nb = ((op >> 3) & 3) + 1
          val = read_csi(nb)
```

```
          val = ("\0" * (4-nb)) + val
          val = val.unpack('N')[0] + delta
          out << [val].pack('N')[4-nb, nb]
        when 0b100, 0b101
          @csinum += n(op, 5, op)
          @csinum &= 0xff
        when 0b110, 0b111
          self.pos += n(op, 5, op)
          self.pos %= csi.size
        end
      end
      out
    end
  end

  if $0 == __FILE__               # test harness
    def decode(s)                 # allow binary and character input
      out = ''
      s.scan(/\s*([01]{8}\s*)|(.)/) do |b, c|
        out << (b ? b.to_i(2).chr : c)
      end
      out
    end
    r = ROHC_RTCP.new
    expect = ''
    DATA.each do |l|
      l.chomp!
      case l
      when /^s (\d+) (.*)$/       # init state item
        r.state_item_is($1.to_i, $2)
      when /^e (.*)$/             # expect result
        expect = decode($1)
      when /^f$/                  # expect failure
        expect = nil
      when /^d (\d+) (.*)$/       # decompress
        refsi = $1.to_i
        testdata = decode($2)
        begin
          out = r.decompress(testdata, refsi)
        rescue DecompressionFailure
          out = nil
        end
        puts "#{testdata.inspect} => #{out.inspect} -- " +
          (expect == out ? 'OK' : "ERROR: Expecting #{expect.inspect}")
      when /^\s*$/, /^#.*$/       # space, comments
      else
        raise "Bad test line #{l}"
      end
```

```
     end
   end
```

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28334
Germany

Phone: +49 421 218 7024
Fax: +49 421 218 7000
Email: cabo@tzi.org

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).