ROHC Implementer’s Guide
<draft-ietf-rohc-rtp-impl-guide-00.txt>

Status of this memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or cite them other than as "work in progress".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/lid-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This document is an individual submission to the IETF. Comments should be directed to the authors.

Abstract

This document describes common misinterpretations and some ambiguous points of ROHC [RFC 3095], which defines the framework and four profiles of a robust and efficient header compression scheme. These points have been identified by the members of the ROHC working group during different interoperability test events.
1. Introduction

ROHC [RFC 3095] addresses a robust and efficient header compression algorithm and as such its description is long and complex. During the implementation and the interoperability tests of the algorithm some unclear areas have been identified. This document tries to collect and clarify these points.

2. CRC Calculation

ROHC uses CRC checksum in order to provide some protection against bit errors. CRC is used in the segmentation protocol and in the compressed packets, as well.

Section 5.2.5.2 describes the segmentation protocol and refers to [RFC 1662], which describes a well-defined CRC algorithm for 32 bit checksums. Although, Section 5.9 only defines the polynomials for 3, 7 and 8-bit long checksum, the same algorithm can be used in these cases as well.

A PERL implementation of the algorithm (written by Carsten Bormann) can be found in Appendix A.
3. Enhanced mode transition procedures

To reduce transmission overhead and computational complexity (including CRC calculation) associated with feedback packets sent for each decompressed packet during mode transition, a decompressor can be implemented with slightly modified mode transition procedures, compared to those defined in [RFC3095].

These modifications affect transitions to Optimistic and Unidirectional modes of operation, i.e. the transitions described in sections 5.6.5 and 5.6.6 of [RFC3095], and make those transition diagrams more consistent with the diagram describing the transition to R-mode. However, the differences between the original diagrams of [RFC3095] were motivated by robustness concerns for mode transitions to Optimistic and Unidirectional modes. To avoid deadlock situations in mode transitions, these aspects must be addressed also when a decompressor implements the enhanced transition procedures, and that is done by following a slightly modified definition of the decompressor transition states. All aspects related to implementation of the enhanced transition procedures are described in subsequent chapters.

Note that these modified operations should be seen only as an improved decompressor implementation, since interoperability is not at all affected. A decompressor implemented according to the optimized procedures would be able to interoperate with an RFC3095 compressor, as well as a decompressor implemented according to the procedures described in RFC3095 would do.

3.1. Modified transition logic for enhanced transitions

The intent with these enhanced transition procedures is to allow the decompressor to stop sending feedback packets for all packets decompressed during the second half of the transition procedure, i.e. after an appropriate IR/IR-DYN/UOR-2 packet has been received from the compressor. In the transition diagrams, sections 3.2 and 3.3 below, this is realized by allowing the decompressor transition parameter (D_TRANS) to be set to P (Pending) at that stage. However, as mentioned above, there are robustness concerns related to this optimization, and to avoid deadlock situations with never completed transitions as a result of feedback losses, the decompressor must continue to send feedback at least periodically, also when in Pending transition state. That would be the equivalence of enhancing the D_TRANS parameter definition in section 5.6.1 of [RFC3095], to include a definition of Pending state operation.

- D_TRANS:
  Possible values for the D_TRANS parameter are (I)NITIATED, (P)ENDING and (D)ONE. D_TRANS MUST be initialized to D, and a mode transition can be initiated only when D_TRANS is D. While D_TRANS is I, the decompressor sends a NACK or ACK carrying a CRC option
for each packet received. When D_TRANS is set to P, the decompressor do not have to send a NACK or ACK for each packet received, but it MUST continue to send feedback on a regular basis, and all feedback packets sent MUST include the CRC option. This ensures that all mode transitions will be completed also in case of feedback losses.

### 3.2. Transition from Reliable to Optimistic mode

The enhanced procedure for transition from Reliable to Optimistic mode is shown below:

<table>
<thead>
<tr>
<th>Compressor</th>
<th>Decompressor</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_TRANS = P</td>
<td>D_TRANS = I</td>
</tr>
<tr>
<td>C_MODE = O</td>
<td></td>
</tr>
<tr>
<td>+---&gt;---&gt;+ I/O-DYN/UOR-2(SN,O)</td>
<td>+---&gt;---&gt;+ D_TRANS = P</td>
</tr>
<tr>
<td>-&lt;-..</td>
<td>D_MODE = O</td>
</tr>
<tr>
<td>-&lt;-..</td>
<td></td>
</tr>
<tr>
<td>ACK(SN,O)</td>
<td>+---&gt;---&gt;+</td>
</tr>
<tr>
<td>+---&gt;---&gt;+</td>
<td>+---&gt;---&gt;+</td>
</tr>
<tr>
<td>UO-0, UO-1*</td>
<td>+---&gt;---&gt;+</td>
</tr>
<tr>
<td>+---&gt;---&gt;+</td>
<td>+---&gt;---&gt;+</td>
</tr>
</tbody>
</table>

### 3.3. Transition to Unidirectional mode

The enhanced procedure for transition to Unidirectional mode is shown on the following figure:
4. Other clarifications

4.1. Padding octet in CRC

According to Section 5.9.1, in case of IR and IR-DYN packets the CRC "is calculated over the entire IR or IR-DYN packet, excluding Payload and including CID or Add-CID octet". Padding isn’t meant to be meaningful part of a packet and not included in CRC calculation. As a result, CRC doesn’t cover an Add-CID octet for CID 0, either.

4.2. Transition to timer-based compression

During transition from window-based compression to timer-based compression it is necessary to keep k large enough to cover both interpretation intervals.

4.3. Generic extension header list

Section 5.7.7.4 defines the static and dynamic parts of the IPv4 header. This section indicates a ‘Generic extension header list’ field in the dynamic chain, which has a variable length. The detailed description of this field can be found in Section 5.8.6.1.

The generic extension header list starts with an octet that is always present, so its length is one octet, at least. If the ‘GP’ bit in the first octet is set to 1 then the length of the list is two octets, even if the list is empty.
4.4. Generic CSRC list

Section 5.7.7.6 defines the static and dynamic parts of the RTP header. This section indicates a ‘Generic CSRC list’ field in the dynamic chain, which has a variable length. This field uses the same encoding rules as the ‘Generic extension header list’ in the IPv4 header, so the same rules apply to its length.

4.5. RTP dynamic chain

Section 5.7.7.6 defines the static and dynamic parts of the RTP header. This section indicates a ‘CC’ field in the dynamic chain. The same field can be found in the ‘Generic CSRC list’ with the same meaning. In order to decode a compressed packet correctly it’s necessary to know the ‘CC’ value because it has serious impact on the packet’s length. In normal case, the values of the fields are equal.

Proposed behavior if the values are different:
Both fields are within the RTP dynamic part but only the second ‘CC’ field resides inside the ‘Generic CSRC list’ together with the XI items. Since the ‘CC’ value determines the number of XI items in the CSRC list and isn’t used otherwise, the first CC field should be ignored and only the second field (inside the CSRC list) should be used for incoming packets. For outgoing packets both fields should be set to the same value.

4.6. Meaning of NBO

In general, an unset flag indicates the normal operation and a set flag indicates unusual behavior. In IPv4 dynamic part (Section 5.7.7.4), if the ‘NBO’ bit is set, it means that network byte order is used. Although, network byte order is the more common byte alignment.

4.7. Implicit TS and IP-ID updates

Type 0 packets (R-0-CRC, UO-0), which contain the compressed RTP sequence number (SN) and a CRC checksum updates not only the RTP sequence number. Such packets also update RTP timestamp and IPv4 ID implicitly, unless there are explicit TS and IP-ID fields in the packet.

4.8. IP-ID

According to Section 5.7 IP-ID means the compressed value of the IPv4 header’s ‘Identification’ field. Type 0, 1 and 2 packets contain the compressed value (IP-ID), however IR packets with dynamic chain and IR-DYN packets transmit the original, uncompressed value. This is because the dynamic part of the IPv4 header (Section 5.7.7.4) contains the ‘Identification’ field instead of the IP-ID.
4.9. Extension-3 in UO-1* packets

Extension-3 is applied to give values and indicate changes to fields other than SN, TS and IP-ID in IP header(s) and RTP header. It is permitted to use it in UO-1* packets, however it may not make any sense because of the updating properties of UO-1* packets (Section 5.7.3).

In case of UO-1* packets, values provided in extensions do not update the context, except those in other SN, TS and IP-ID fields. The SN, TS and IP-ID fields can also be sent in Extension-0, -1 and -2 and the other fields of Extension-3 don’t update the context.

4.10. Extension-3 in UOR-2* packets

If Extension-3 is used in a UOR-2* packet then the information of the extension updates the context. Some flags of the IP header in the extension (e.g. NBO or RND) can change the interpretation of fields in UOR-2* packets.

In these cases, when a flag changes in Extension-3, a decompressor should re-parse the UOR-2* packet.

4.11. Multiple SN options in one feedback packet

The length of the sequence number field in the original RTP header is 32 bits. A decompressor can’t send back all the 32 bits in a feedback packet unless it uses multiple SN options in one feedback packet. Section 5.7.6.1 declares that a FEEDABCK-2 packet can contain variable number of feedback options and the options can appear in any order.

A compressor – that wants to be conform to the specification – should be able to process multiple SN options in one feedback packet.

4.12. Packet decoding during mode transition

Each ROHC profile defines its own set of packet formats, and also its own feedback packets. The use of various operational modes is also defined by each specific profile. A decompressor can therefore not initiate a mode transfer request before at least one packet of a new context has been correctly decompressed, establishing the context based on one specific profile (as specified in IR packets). First then the context has been established, the decompressor knows the profile used, which modes are defined by that profile, and the feedback packet formats available.

If the transition procedures in sections 5.6.5 and 5.6.6 of [RFC3095] are followed (and not the enhanced procedures described in section 3 of this document), it is important to note that type 0 or type 1
packets may be received by the decompressor during the first half of the transition procedure, and these packets must not mistakenly be interpreted as the packets sent by the compressor to indicate completed transition. The decompressor side must therefore keep track of the transition status, e.g. with an additional parameter. If the enhanced transition procedures described in section 3 of this document are used, the D_TRANS parameter can serve this purpose since its definition and usage is slightly modified.

5. Test Configuration

ROHC is used to compress IP/UDP/RTP, IP/UDP and IP/ESP headers, thus every ROHC implementation has an interface that can send and receive IP packets (i.e. Ethernet). On the other hand, there must be an interface (a serial link for example) or other means of transport (an IP/UDP flow), which can transmit ROHC packets. Having these two interfaces several configurations can be set up. The figure below shows sample configurations that can be used for testing a ROHC implementation:

![Sample Configuration Diagram]

Unfortunately, comparing the IP/UDP/RTP packets at the endpoints can only show whether the reconstructed stream differs from the original or not. In order to identify the place of the error more detailed tests are needed. The next figure shows another possible scenario:

![Another Sample Configuration Diagram]

In the first case, the test equipment generates the RTP stream and also acts as a ROHC decompressor. The tester must recognize if the original RTP stream was compressed in a bad way and gives an alarm. In the second case, it is the test equipment that sends the compressed ROHC packets and the Decompressor reconstructs the RTP stream. Since the tester knows the ROHC packets and the reconstructed RTP stream it can detect if the Decompressor makes a mistake.

6. References
[RFC-3095] C. Bormann, Editor, ROBust Header Compression (ROHC),

[RFC-1662] W. Simpson, Editor, PPP in HDLC-like Framing, RFC 1662,

7. Authors’ Addresses

Peter Kremer
Ericsson Research, Conformance and Software Test Laboratory
H-1300 Bp. 3., P.O. Box 107, HUNGARY

Phone: +36-1-437-7033
Fax: +36-1-437-7767
Email: Peter.Kremer@ericsson.com

Lars-Erik Jonsson
Box 920
Ericsson Erisoft AB
SE-971 28 Lulea, Sweden

Phone: +46 920 20 21 07
Fax: +46 920 20 20 99
EMail: lars-erik.jonsson@ericsson.com
Appendix A. Sample CRC algorithm

#!/usr/bin/perl -w
use strict;
#=================================
# ROHC CRC demo - Carsten Bormann cabo@tzi.org 2001-08-02
# This little demo shows the three types of CRC in use in RFC 3095,
# the robust header compression standard. Type your data in
# hexadecimal form and the press Control+D.
#---------------------------------
# utility
#
# sub dump_bytes($) {  
#   my $x = shift;
#   my $i;
#   for ($i = 0; $i < length($x); ) {  
#     printf("%02x ", ord(substr($x, $i, 1)));
#     printf("\n") if (++$i % 16 == 0);
#   }
#   printf("\n") if ($i % 16 != 0);
# }
#
#---------------------------------
# The CRC calculation algorithm.
#
# sub do_crc($$$) {  
#   my $nbits = shift;
#   my $poly = shift;
#   my $string = shift;
#   my $crc = ($nbits == 32 ? 0xffffffff : (1 << $nbits) - 1);
#   for (my $i = 0; $i < length($string); ++$i) {  
#     my $byte = ord(substr($string, $i, 1));
#     for( my $b = 0; $b < 8; $b++ ) {  
#       if (($crc & 1) ^ ($byte & 1)) {  
# $crc >>= 1;
# $crc ^= $poly;
# } else {  
# $crc >>= 1;
#   }  
# $byte >>= 1;
#   }
#   printf "%2d bits, ", $nbits;
#   printf "CRC: %02x\n", $crc;
# }
#---------------------------------
#  Test harness
# $
=/ = undef;
$/_ = <>;  # read until EOF
my $string = ""; # extract all that looks hex:
s/([0-9a-fA-F][0-9a-fA-F])/$string .= chr(hex($1)), ""/eg;
dump_bytes($string);
#---------------------------------

# 32-bit segmentation CRC
# Note that the text implies this is complemented like for PPP
# (this differs from 8, 7, and 3-bit CRC)
#      C(x) = x^0 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^10 +
#            x^11 + x^12 + x^16 + x^22 + x^23 + x^26 + x^32
# do_crc(32, 0xedb88320, $string);

#---------------------------------
# 8-bit IR/IR-DYN CRC
#      C(x) = x^0 + x^1 + x^2 + x^8
# do_crc(8, 0xe0, $string);

#---------------------------------
# 7-bit FO/SO CRC
#      C(x) = x^0 + x^1 + x^2 + x^3 + x^6 + x^7
# do_crc(7, 0x79, $string);

#---------------------------------
# 3-bit FO/SO CRC
#      C(x) = x^0 + x^1 + x^3
# do_crc(3, 0x6, $string);