

Robust Header Compression
Internet-Draft
Expires: April 18, 2005

E. Brinkmann
C. Bormann
Universitaet Bremen TZI
October 18, 2004

RTP-ROHC in ROHC Formal Notation
draft-brinkmann-rohc-3095-fn-00.txt

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of [section 3 of RFC 3667](#). By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she become aware will be disclosed, in accordance with [RFC 3668](#).

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 to 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 Internet-Draft will expire on April 18, 2005.

Copyright Notice

Copyright (C) The Internet Society (2004).

Abstract

[RFC 3095](#) [2] defines four ROHC profiles for the header compression of IP, UDP, RTP, ESP, and related protocol headers. [RFC 3095](#) is defined in English language and traditional RFC box notation. The present document gives a definition of [RFC 3095](#)'s packet formats in the ROHC Formal Notation that was defined to facilitate the development of the TCP header compression ROHC profile.

Internet-Draft

3095-FN

October 2004

\$Revision: 1.10 \$

Table of Contents

1.	Introduction	3
2.	RFC 3095 Formal Notation Specification	3
3.	References	21
	Authors' Addresses	22
	Intellectual Property and Copyright Statements	23

Internet-Draft

3095-FN

October 2004

1. Introduction

[RFC 3095](#) [2] defines four ROHC profiles for the header compression of IP, UDP, RTP, ESP, and related protocol headers. [RFC 3095](#) is defined in English language and traditional RFC box notation.

ROHC FN [1] is a formal notation designed to help with the creation of new ROHC [RFC-3095](#) [2] header compression profiles, in particular the TCP ROHC [3] profile currently under development.

The present document gives a definition of [RFC 3095](#)'s packet formats in the ROHC Formal Notation defined for the TCP header compression ROHC profile.

This work is intended to serve three purposes:

- o validate ROHC-FN further by showing that large parts of [RFC 3095](#) can be reasonably specified in it,
- o facilitate work on automatic validation and test case generation for [RFC 3095](#), and
- o possibly contribute to a Draft Standard version of [RFC 3095](#) and related profiles (LLA [4], R-mode LLA [5], IP ROHC [6], and UDP-lite based RTP ROHC [7]), in case it is decided to use ROHC-FN to facilitate advancing these documents to Draft Standards level.

A number of caveats apply to the current version of the specification:

- o Formats are currently defined for profile 0x0001 (RTP) only. The other [RFC 3095](#) profiles, as well as the profiles for the related standards, will be added once the profile 0x0001 specification stabilizes.
- o The current specification is based on an interim state of the ROHC-FN notation. This interim state has significant changes from the -03 version [1] referenced above. Further changes might occur on the way to the -04 version of the notation, which cannot be tracked in this document because a later draft deadline applies to

- the notation.
- o There are still a few open issues. In particular, the way list notation can be used to represent [RFC 3095](#) list compression is not yet entirely worked out. Further open issues include the handling of GRE and authentication header data and the encoding of extension formats. Finally, the extent to which the expressive power of the notation should be used to express the english language constraints noted with the individual packet formats needs to be decided. See the FIXMEs below for details.

2. [RFC 3095](#) Formal Notation Specification

```
%% RFC 3095 Profile 0x0001 (RTP/UDP/IP) in ROHC-FN
% $Revision: 1.20 $
```

```
% TODO:
```

```
%
```

```
% - handle AH and GRE
```

```
% - extension formats
```

```
% - CSRC list (currently defined as static, must use list encoding)
```

```
% - further conditions for selection / disambiguation between different
% packet formats in rtp_udp_ip structure (for R-1*, U0-1* and UOR-2*,
% see comments below and notes in RFC 3095)
```

```
% - better description for encoding methods not defined in FN
```

```
% (RTP timestamp encoding and others)
```

```
% Notes:
```

```
%
```

```
% - This specification uses the field attributes 'context_value' and
% 'context_length' which are not defined in the current FN draft
% to refer to a field's context value.
```

```
%%
%
```

```
% Encoding methods used in this profile that are not defined in FN:
```

```
ah_data(ah) == "Encoding of authentication header data according to
section 5.8.4.2 of RFC 3095";
```

% FIXME: AH handling needs some extensions to decide at which
% place in the compressed header the data appear (depending on
% the conditions specified in 5.8.4.2).

inferred_ipv4_length == "This encoding method does not generate any bits
in the compressed header. Upon decompression the uncomp_value
attribute of the IPv4 total length field is inferred from the
amount of uncompressed data.";

inferred_ipv6_length == "This encoding method does not generate any bits
in the compressed header. Upon decompression the uncomp_value
attribute of the IPv6 payload length field is inferred from
the amount of uncompressed data.";

inferred_udp_length == "This encoding method does not generate any bits
in the compressed header. Upon decompression the uncomp_value
attribute of the UDP length field is inferred from the amount
of uncompressed data.";

inferred_ipv4_header_checksum == "This encoding method does not generate

any bits in the compressed header. Upon decompression the
checksum value is re-calculated from the uncompressed header
data as specified in [section 3.1 of RFC 791](#).";

inferred_rtp_timestamp == "This encoding method does not generate any
bits in the compressed header. Upon decompression the
uncompressed timestamp value is calculated from the RTP
sequence number.";

% FIXME: Detailed description / reference

rtp_timestamp(bits) == "Compresses RTP timestamps as described in
[section 5.7 of RFC 3095](#)";

%

% If value(TIME_STRIDE) > 0, timer-based compression of the RTP
% Timestamp is used (see [section 4.5.4](#)).

%

% If value(Tsc) = 1, Scaled RTP Timestamp encoding is used before
% compression (see [section 4.5.3](#)), and default-slope(TS) = 1.

%

% If value(Tsc) = 0, the Timestamp value is compressed as-is, and
% default-slope(TS) = value(TS_STRIDE).

```

%
% The interpretation intervals, see section 4.5.1, are defined as
% follows:
%
%      p = 2^(bits(TS)-2) - 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Misc Stuff
%

crc3 (data_value, data_length) ===
{
    uncompressed_format = ;

    compressed_format = crc_value, %[ 3 ]
    {
        crc_value ::= crc(3, 0x06, 0x07, data_value, data_length);
    };
};

crc7 (data_value, data_length) ===
{
    uncompressed_format = ;

    compressed_format = crc_value, %[ 7 ]

```

```

{
    crc_value ::= crc(7, 0x79, 0x7f, data_value, data_length);
};
};

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% IPv4 Header
%

```

```

ipv4 (id, rnd, nbo) === {

    uncompressed_format = version,      %[ 4 ]

```

```

hdr_length, %[ 4 ]
tos,         %[ 6 ]
ecn_flags,   %[ 2 ]
length,      %[ 16 ]
ip_id,       %[ 16 ]
df,          %[ 1 ]
mf,          %[ 1 ]
rf,          %[ 1 ]
offset,      %[ 13 ]
ttl,         %[ 8 ]
protocol,    %[ 8 ]
checksum,    %[ 16 ]
src_addr,    %[ 32 ]
dst_addr;    %[ 32 ]

```

```

default_methods =
{

```

```

    % FIXME: The fields ip_id_rnd and ip_id_nbo are flags that
    % are NOT present in the compressed or uncompressed IPv4
    % header. They should be bound with the RND and NBO flags
    % corresponding to this IPv4 header.
    %
    let (ip_id_rnd:uncomp_length == 1);
    let (ip_id_nbo:uncomp_length == 1);

    version      ::= uncompressed_value(4, 4);
    hdr_length   ::= uncompressed_value(4, 5);
    tos          ::= static;
    ecn_flags    ::= static;
    length       ::= inferred_ipv4_length;
    ip_id        ::= irregular(16);
    ip_id_rnd    ::= static;
    ip_id_nbo    ::= static;
    df           ::= static;

```

```

mf           ::= uncompressed_value(1, 0);
rf           ::= static;
offset       ::= uncompressed_value(13, 0);
ttl          ::= static;
protocol     ::= static; % in RTP/UDP/IP profile always 17 (UDP)
checksum     ::= inferred_ipv4_header_checksum;
src_addr     ::= static;

```

```

        dst_addr    ::= static;
    };

    format_ipv4 =
    {
        let (id == ip_id:uncomp_value);
        let (rnd == ip_id_rnd:uncomp_value);
        let (nbo == ip_id_nbo:uncomp_value);
    };

};

ip_id_offset (nbits, sn, id1, rnd1, nbo1, id2, rnd2, nbo2) == {
    % id1 (+ rnd1, nbo1) = IP-ID of inner IP header
    % id2 (+ rnd2, nbo2) = IP-ID of outer IP header

    % This encoding method will succeed only when rnd (which
    % should be the RND or RND2 flag) is 0, i.e. packet formats
    % using this method are not applicable to packet streams with
    % random IP-IDs.

    uncompressed_format = ;

    format_inner_nbo = ip_id,
    {
        let (rnd1 == 0);
        let (nbo1 == 1);

        let (ip_id:uncomp_value == id1 - sn);
        let (ip_id:uncomp_length == 16);

        ip_id ::= lsb(nbits, 0);
    };

    format_inner_non_nbo = ip_id,
    {
        let (rnd1 == 0);
        let (nbo1 == 0);

        let (ip_id:uncomp_value ==

```



```

        ((id1 / 256) + (id1 & 255) * 256) - sn);
    let (ip_id:uncomp_length == 16);

    ip_id ::= lsb(nbits, 0);
};

format_outer_nbo = ip_id,
{
    let (rnd1 == null || rnd1 == 1);
    let (rnd2 == 0);
    let (nbo2 == 1);

    let (ip_id:uncomp_value == id2 - sn);
    let (ip_id:uncomp_length == 16);

    ip_id ::= lsb(nbits, 0);
};

format_outer_non_nbo = ip_id,
{
    let (rnd1 == null || rnd1 == 1);
    let (rnd2 == 0);
    let (nbo2 == 0);

    let (ip_id:uncomp_value ==
        ((id2 / 256) + (id2 & 255) * 256) - sn);
    let (ip_id:uncomp_length == 16);

    ip_id ::= lsb(nbits, 0);
};

};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% IPv6 Header
%

ipv6 == {

    uncompressed_format = version,          %[ 4 ]
                        traffic_class,      %[ 6 ]
                        ecn_flags,         %[ 2 ]
                        flow_label,        %[ 20 ]
                        payload_length,    %[ 16 ]
                        next_header,       %[ 8 ]
                        hop_limit,         %[ 8 ]

```

Internet-Draft

3095-FN

October 2004

```
src_addr,      %[ 128 ]
dst_addr;      %[ 128 ]

default_methods =
{
    version      ::= uncompressed_value(4, 6);
    traffic_class ::= static;
    ecn_flags     ::= static;
    flow_label    ::= static;
    payload_length ::= inferred_ipv6_length;
    next_header   ::= static;
    hop_limit     ::= static;
    src_addr      ::= static;
    dst_addr      ::= static;
};

compressed_format = ;

};

%
% IP Headers
%
%
% IP Headers (id1, rnd1, nbo1, id2, rnd2, nbo2) == {
% id1 (+ rnd1, nbo1) = IP-ID of inner IP header
% id2 (+ rnd2, nbo2) = IP-ID of outer IP header

uncompressed_format_ipv4 = ipv4_header;

uncompressed_format_ipv6 = ipv6_header;

uncompressed_format_ipv4v4 = outer_ipv4_header,
                             inner_ipv4_header;

uncompressed_format_ipv4v6 = outer_ipv4_header,
                             inner_ipv6_header;

uncompressed_format_ipv6v4 = outer_ipv6_header,
                             inner_ipv4_header;
```

```
uncompressed_format_ipv6v6 = outer_ipv6_header,  
                               inner_ipv6_header;
```

```
format_ipv4 =
```

```
{  
    ipv4_header ::= ipv4(id1, rnd1, nbo1);  
    let (id2 == null);  
    let (rnd2 == null);  
    let (nbo2 == null);  
};  
  
format_ipv6 =  
{  
    ipv6_header ::= ipv6;  
    let (id1 == null);  
    let (rnd1 == null);  
    let (nbo1 == null);  
    let (id2 == null);  
    let (rnd2 == null);  
    let (nbo2 == null);  
};  
  
format_ipv4v4 =  
{  
    outer_ipv4_header ::= ipv4(id2, rnd2, nbo2);  
    inner_ipv4_header ::= ipv4(id1, rnd1, nbo1);  
};  
  
format_ipv4v6 =  
{  
    outer_ipv4_header ::= ipv4(id2, rnd2, nbo2);  
    inner_ipv6_header ::= ipv6;  
    let (id1 == null);  
    let (rnd1 == null);  
    let (nbo1 == null);  
};  
  
format_ipv6v4 =  
{  
    outer_ipv6_header ::= ipv6;
```



```

};                                     % side of it only:

format_compressed =      % NOTE: udp_checksum not present here!
{
    let (udp_checksum:uncomp_length == 16);
    let (udpcs == udp_checksum:uncomp_value);
    let (udp_checksum:context_length == 16);
    let (udpcs_ctx == udp_checksum:context_value);
};

};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Extension formats (section 5.7.5 of RFC 3095)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

extension (x) == {

    uncompressed_format = ;

    format_without_extension =
    {
        let (x == 0);
    };

    % FIXME: The following four extension formats do not yet contain
    % the encodings for their fields. These will need several additional
    % structures (in particular for the optional fields) to be defined.
    % A number of values will have to be passed as additional parameters
    % to this structure as their uncompressed side is defined in other
    % structures.

    format_extension0 = discriminator, %[ 2 ]
                        sn,             %[ 3 ]
                        pt,             %[ 3 ]
    {
        let (x == 1);
        discriminator ::= '00';
    }
}

```

```

};

format_extension1 = discriminator, %[ 2 ]
                    sn,             %[ 3 ]
                    pt,             %[ 3 ]
                    mt,             %[ 8 ]
{
    let (x == 1);
    discriminator ::= '01';
};

format_extension2 = discriminator, %[ 2 ]
                    sn,             %[ 3 ]
                    pt,             %[ 11 ]
                    mt,             %[ 8 ]
{
    let (x == 1);
    discriminator ::= '10';
};

format_extension3 = discriminator, %[ 2 ]
                    s,              %[ 1 ]
                    r_ts,           %[ 1 ]
                    tsc,            %[ 1 ]
                    i,              %[ 1 ]
                    ip,             %[ 1 ]

```

```

                    rtp,            %[ 1 ]
                    inner_ip_flags, % 0 or 7 bits
                    ip2,            % 0 or 1 bit
                    outer_ip_flags, % 0 or 8 bits
                    sn,             % 0 or 8 bits
                    ts,             % 0 to 4 octets
                    inner_ip_fields, % optional, variable length
                    ip_id,          % 0 or 16 bits
                    outer_ip_fields, % optional, variable length
                    rtp_flags_and_fields, % optional, variable length
{
    let (x == 1);
    discriminator ::= '11';
};

```

```
};
```

```
%%%%%%%%%%  
%  
% Random Parts  
%  
%%%%%%%%%
```

```
random_ip_id (id, rnd) == {  
  
    uncompressed_format = ;  
  
    format_without_ip_id =  
    {  
        let (rnd == 0);  
    };  
  
    format_with_ip_id = ip_id,  
    {  
        let (rnd == 1);  
        let (id == ip_id:uncomp_value);  
        ip_id ::= irregular(16);  
    };  
  
};
```

```
random_ah_data (ah) == {  
  
    uncompressed_format = ;  
  
    format_without_ah_data =
```

```
{  
    let (ah == null);  
};  
  
format_with_ah_data = ah_data,  
{  
    let (ah != null);  
    ah_data ::= ah_data(ah);
```

```

    };

};

random_gre_checksum (gre) === {

    uncompressed_format = ;

    format_without_gre_checksum =
    {
        let (gre == null);
    };

    format_with_gre_checksum = gre_checksum,
    {
        let (gre != null);
        let (gre == gre_checksum:uncomp_value);
        gre_checksum ::= irregular(16);
    };

};

random_ip_data (id, rnd, ah, gre) === {

    uncompressed_format = ;

    compressed_format = ip_id,
                        ah_data,
                        gre_checksum,
    {
        ip_id          ::= random_ip_id(id, rnd);
        ah_data         ::= random_ah_data(ah);
        gre_checksum    ::= random_gre_checksum(gre);
    };

};

```

```

random_udp_checksum (udpcs, udpcs_ctx) === {

```



```

uncompressed_format = ;

format_without_udp_checksum =
{
    let (udpcs_ctx == 0);
    let (udpcs == 0);
};

format_with_udp_checksum = udp_checksum,
{
    let (udpcs_ctx != 0);
    let (udpcs == udp_checksum:uncomp_value);
    udp_checksum ::= irregular(16);
};

};

random_parts (id1, rnd1, id2, rnd2,
              ah1, ah2, gre1, gre2, udpcs, udpcs_ctx) == {

    uncompressed_format = ;

    compressed_format = outer_ip,
                        inner_ip,
                        udp_checksum,
    {
        outer_ip      ::= random_ip_data(id1, rnd1, ah1, gre1);
        inner_ip      ::= random_ip_data(id2, rnd2, ah2, gre2);
        udp_checksum  ::= random_udp_checksum(udpcs, udpcs_ctx);
    };

};

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% RTP/UDP/IP Header
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R = 0;      % compression modes
U = 1;
O = 2;

```

```
rtp_udp_ip == {
```

```
    uncompressed_format = ip_headers,  
                           udp_header, %[ 8 ]  
                           version,    %[ 2 ]  
                           padding,    %[ 1 ]  
                           rtpext,     %[ 1 ]  
                           csrc_count, %[ 4 ]  
                           marker,     %[ 1 ]  
                           payload,    %[ 7 ]  
                           seq_number, %[ 16 ]  
                           timestamp,  %[ 32 ]  
                           ssrc,       %[ 32 ]  
                           csrc;
```

```
    default_methods =  
    {
```

```
        % FIXME: The mode field represents the current mode of  
        % operation. This binding is not yet specified by means  
        % of the FN syntax.
```

```
        let (mode:uncomp_length == 2);
```

```
        mode          ::= static;
```

```
        %
```

```
        ip_headers    ::= ip_headers(id1, rnd1, nbo1,  
                                     id2, rnd2, nbo2);
```

```
        %
```

```
        udp_header    ::= udp_header(udpcs, udpcs_ctx);
```

```
        %
```

```
        version       ::= uncompressed_value(2, 2);
```

```
        padding       ::= static;
```

```
        rtpext        ::= static;
```

```
        csrc_count    ::= static;
```

```
        marker        ::= static;
```

```
        payload       ::= static;
```

```
        timestamp     ::= inferred_rtp_timestamp;
```

```
        ssrc          ::= static;
```

```
        csrc          ::= static;
```

```
        %
```

```
        x             ::= irregular(1);
```

```
        extension     ::= extension(x);
```

```
        %
```

```
        % FIXME: AH data and GRE checksums
```

```
        random_parts  ::= random_parts(id1, rnd1,  
                                     id2, rnd2,  
                                     null, null,
```

```
                                     % AH data 1 + 2
```

```

null, null,          % GRE checksums
udpcs, udpcs_ctx);

```

Internet-Draft

3095-FN

October 2004

} ;

0%
0%

5.7.1. Packet type 0: U0-0, R-0, R-0-CCR

```
format_r_0 = discriminator, %[ 2 ]
            seq_number,    %[ 6 ]
            random_parts,
```

```
{
    let (mode:uncomp_value == R);

    discriminator ::= '00';
    seq_number    ::= lsb(6, 1);
};
```

```
format_r_0_crc = discriminator, %[ 2 ]
                  seq_number,    %[ 7 ]
                  crc,           %[ 7 ]
                  random_parts,
```

```
{
    let (mode:uncomp_value == R);

    discriminator ::= '01';
    seq_number    ::= lsb(7, 3);
    crc           ::= crc7(this:uncomp_value, this:uncomp_length);
};
```

```
format_uo_0 = discriminator, %[ 1 ]
              seq_number,    %[ 4 ]
              crc,           %[ 3 ]
              random_parts,
```

```
{
    let (mode:uncomp_value == U || mode:uncomp_value == 0);

    discriminator ::= '0';
    seq_number    ::= lsb(4, 1);
    crc          ::= crc3(this:uncomp_value, this:uncomp_length);
}
```

```
};
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
```

```
5.7.2. Packet type 1 (R-mode): R-1, R-1-TS, R-1-ID
```

```
format_r_1 = discriminator, %[ 2 ]  
            seq_number,    %[ 6 ]
```

```
        marker,          %[ 1 ]  
        x,                %[ 1 ]  
        timestamp,       %[ 6 ]  
        extension,  
        random_parts,  
    {  
        let (mode:uncomp_value == R);  
        % FIXME: context MUST NOT contain IPv4 headers with value(RND)  
  
        discriminator ::= '10';  
        seq_number    ::= lsb(6, 1);  
        marker        ::= irregular(1);  
        timestamp     ::= rtp_timestamp(6);  
    };  
  
    format_r_1_id = discriminator, %[ 2 ]  
                  seq_number,    %[ 6 ]  
                  marker,        %[ 1 ]  
                  x,              %[ 1 ]  
                  t,              %[ 1 ]  
                  ip_id,         %[ 5 ]  
                  extension,  
                  random_parts,  
    {  
        let (mode:uncomp_value == R);  
        % FIXME: context MUST contain at least one IPv4 header  
        let (rnd1 == 0 || rnd2 == 0);  
  
        discriminator ::= '10';  
        seq_number    ::= lsb(6, 1);  
        marker        ::= irregular(1);  
        t              ::= '0';
```

```

        ip_id            ::= ip_id_offset(5, seq_number:uncomp_value,
                                           id1, rnd1, nbo1,
                                           id2, rnd2, nbo2);
    };

    format_r_1_ts = discriminator, %[ 2 ]
                    seq_number,      %[ 6 ]
                    marker,          %[ 1 ]
                    x,               %[ 1 ]
                    t,               %[ 1 ]
                    timestamp,       %[ 5 ]
                    extension,
                    random_parts,
    {
        let (mode:uncomp_value == R);    % FIXME: see note in 3095
        discriminator ::= '10';
    }

```

```

        seq_number      ::= lsb(6, 1);
        marker          ::= irregular(1);
        t               ::= '1';
        timestamp       ::= rtp_timestamp(5);
    };

```

%%%

%

5.7.3. Packet type 1 (U/0-mode): U0-1, U0-1-ID, U0-1-TS

```

    format_uo_1 = discriminator, %[ 2 ]
                  timestamp,     %[ 6 ]
                  marker,        %[ 1 ]
                  seq_number,    %[ 4 ]
                  crc,           %[ 3 ]
                  random_parts,
    {
        let (mode:uncomp_value == U || mode:uncomp_value == 0);
        % FIXME: see note in 3095

        discriminator ::= '10';
        timestamp      ::= rtp_timestamp(6);
        marker         ::= irregular(1);
        seq_number     ::= lsb(4, 1);
    }

```

```

        crc                ::= crc3(this:uncomp_value, this:uncomp_length);
};

format_uo_1_id = discriminator, %[ 2 ]
                t,                %[ 1 ]
                ip_id,            %[ 5 ]
                x,                %[ 1 ]
                seq_number,       %[ 4 ]
                crc,              %[ 3 ]
                extension,
                random_parts,
{
    let (mode:uncomp_value == U || mode:uncomp_value == 0);
    % FIXME: see note in 3095

    discriminator ::= '10';
    t              ::= '0';
    ip_id          ::= ip_id_offset(5, seq_number:uncomp_value,
                                     id1, rnd1, nbo1,
                                     id2, rnd2, nbo2);

    seq_number     ::= lsb(4, 1);
    crc            ::= crc3(this:uncomp_value, this:uncomp_length);
};

```

```

format_uo_1_ts = discriminator, %[ 2 ]
                t,                %[ 1 ]
                timestamp,        %[ 5 ]
                marker,           %[ 1 ]
                seq_number,       %[ 4 ]
                crc,              %[ 3 ]
                random_parts,
{
    let (mode:uncomp_value == U || mode:uncomp_value == 0);
    % FIXME: see note in 3095

    discriminator ::= '10';
    t              ::= '1';
    timestamp      ::= rtp_timestamp(5);
    marker         ::= irregular(1);
    seq_number     ::= lsb(4, 1);
    crc            ::= crc3(this:uncomp_value, this:uncomp_length);
};

```



```

        crc          ::= crc7(this:uncomp_value, this:uncomp_length);
    };

    format_uor_2_ts = discriminator, %[ 3 ]
                    timestamp,      %[ 5 ]
                    t,              %[ 1 ]
                    marker,         %[ 1 ]
                    seq_number,     %[ 6 ]
                    x,              %[ 1 ]
                    crc,            %[ 7 ]
                    extension,
                    random_parts,
    {
        % FIXME: restrictions (see note in 3095)
        discriminator ::= '110';
        timestamp     ::= rtp_timestamp(5);
        t             ::= '1';
        marker        ::= irregular(1);
        seq_number    ::= lsb(6, 1);
        crc           ::= crc7(this:uncomp_value, this:uncomp_length);
    };
};

```

3 References

- [1] Price, R., "Formal Notation for Robust Header Compression (ROHC-FN)", [draft-ietf-rohc-formal-notation-03](#) (work in progress), July 2004.
- [2] 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.

- [3] Pelletier, G., "RObust Header Compression (ROHC): Profile for TCP/IP (ROHC-TCP)", [draft-ietf-rohc-tcp-07](#) (work in progress),

July 2004.

- [4] Jonsson, L-E. and G. Pelletier, "RObust Header Compression (ROHC): A Link-Layer Assisted Profile for IP/UDP/RTP", [RFC 3242](#), April 2002.
- [5] Liu, Z. and K. Le, "Zero-byte Support for Bidirectional Reliable Mode (R-mode) in Extended Link-Layer Assisted RObust Header Compression (ROHC) Profile", [RFC 3408](#), December 2002.
- [6] Jonsson, L-E. and G. Pelletier, "RObust Header Compression (ROHC): A Compression Profile for IP", [RFC 3843](#), June 2004.
- [7] Pelletier, G., "RObust Header Compression (ROHC): Profiles for UDP-Lite", [draft-ietf-rohc-udp-lite-04](#) (work in progress), June 2004.

Authors' Addresses

Eilert Brinkmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28334
Germany

Phone: +49 421 218 7845
Fax: +49 421 218 7000
EMail: eilert@tzi.org

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

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

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

