Internet Engineering Task Force (IETF)
Request for Comments: 6262
Category: Standards Track
ISSN: 2070-1721

S. Ikonin
SPIRIT DSP

August 2011

RTP Payload Format for IP-MR Speech Codec

Abstract

This document specifies the payload format for packetization of SPIRIT IP-MR encoded speech signals into the Real-time Transport Protocol (RTP). The payload format supports transmission of multiple frames per packet and introduces redundancy for robustness against packet loss and bit errors.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6262.

Table of Contents

1. Introduction

This document specifies the payload format for packetization of SPIRIT IP-MR encoded speech signals into the Real-time Transport Protocol (RTP). The payload format supports transmission of multiple frames per packet and introduces redundancy for robustness against packet loss and bit errors.

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 [RFC2119].

2. IP-MR Codec Description

IP-MR is a wideband speech codec designed by SPIRIT for conferencing services over packet-switched networks such as the Internet.

IP-MR is a scalable codec. This means that the source not only has the ability to change transmission rate on the fly, but the gateway is also able to decrease bandwidth at any time without performance overhead. There are 6 coding rates from 7.7 to 34.2 kbps available.

The codec operates on a frame-by-frame basis with a frame size of 20 ms at a 16 kHz sampling rate with a total end-to-end delay of 25 ms. Each compressed frame is represented as a sequence of layers. The first (base) layer is mandatory while the other (enhancement) layers can be safely discarded. Information about the particular frame structure is available from the payload header. In order to adjust outgoing bandwidth, the gateway MUST read the frame(s) structure from the payload header, define which enhancement layers to discard, and compose a new RTP packet according to this specification.

In fact, not all bits within a frame are equally tolerant to distortion. IP-MR defines 6 classes ('A'-'F') of sensitivity to bit errors. Any damage of class 'A' bits causes significant reconstruction artifacts while the loss in class 'F' may not even be perceived by the listener. Note that only the base layer in a bitstream is represented as a set of classes.

The IP-MR payload format allows frame duplication through the packets to improve robustness against packet loss (Section 3.6). The base layer can be retransmitted completely or in several sensitive classes. Enchantment layers are not retransmittable.

The fine-grained redundancy in conjunction with bitrate scalability allows applications to adjust the trade-off between overhead and robustness against packet loss. Note that this approach is supported natively within a packet and requires no out-of-band signals or session-initialization procedures.

The main IP-MR features are as follows:

- High-quality wideband speech codec.
- Bitrate scalable with 6 average rates from 7.7 to 34.2 kbps.
- Built-in discontinuous transmission (DTX) and comfort noise generation (CNG) support.

- Flexible in-band redundancy control scheme for packet-loss protection.

3. Payload Format

The payload format consists of the RTP header and the IP-MR payload.

3.1. RTP Header Usage

The format of the RTP header is specified in [RFC3550]. This payload format uses the fields of the header in a manner consistent with that specification.

The RTP timestamp corresponds to the sampling instant of the first sample encoded for the first frame-block in the packet. The timestamp clock frequency SHALL be 16 kHz. The duration of one frame is 20 ms, which corresponds to 320 samples per frame. Thus, the timestamp is increased by 320 for each consecutive frame. The timestamp is also used to recover the correct decoding order of the frame-blocks.

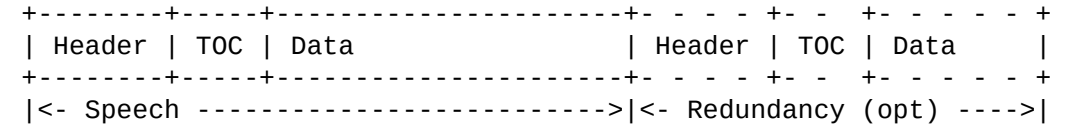
The RTP header marker bit (M) SHALL be set to 1 whenever the first frame-block carried in the packet is the first frame-block in a talkspurt (see definition of talkspurt in Section 4.1 of [RFC3551]). For all other packets, the marker bit SHALL be set to zero (M=0).

The assignment of an RTP payload type for the format defined in this memo is outside the scope of this document. The RTP profiles in use currently mandate binding the payload type dynamically for this payload format. This is basically necessary because the payload type expresses the configuration of the payload itself, i.e., basic or interleaved mode, and the number of channels carried.

The remaining RTP header fields are used as specified in [RFC3550].

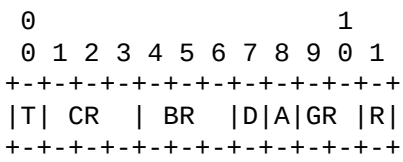
3.2. RTP Payload Structure

The IP-MR payload is composed of two payloads, one for current speech and one for redundancy. Both payloads are represented in this form: Header, Table of Contents (TOC), and Data. Redundancy payload carries data for preceding and pre-preceding packets.



3.3. Speech Payload Header

This header carries parameters that are common for all frames in the packet:



- T (1 bit): Reserved. MUST always be set to 0. Receiver MAY discard packet if the 'T' bit is not equal to 0.

- CR (3 bits): Coding rate index - top enchantment layer available. The CR value 7 (NO_DATA) indicates that there is no speech data (and thus no speech TOC) in the payload. This MAY be used to transmit redundancy data only.

- BR (3 bits): Base rate index - base layer bitrate. Speech payload can be scaled to any rate index between BR and CR. Packets with BR = 6 or BR > CR MUST be discarded. Redundancy data is also considered to have a base rate of BR.

- D (1 bit): Reserved. MUST always be set to 1. Receiver MAY discard packet if the 'D' bit is zero.

- A (1 bit): Byte alignment. The value of 1 specifies that padding bits were added to enable each compressed frame (3.5) to start with the byte (8-bit) boundary. The value of 0 specifies unaligned frames. Note that the speech payload is always padded to the byte boundary independently on an 'A' bit value.

- GR (2 bits): Number of frames in packet (grouping size). Actual grouping size is GR + 1; thus, the maximum grouping supported is 4.

- R (1 bit): Redundancy presence. Value of 1 indicates redundancy payload presence.

Note that the values of 'T' and 'D' bits are fixed; any other values are not allowed by specification. Padding bits ('P' bits) MUST always be set to zero.

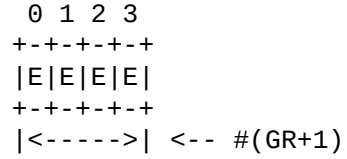
The following table defines the mapping between rate index and rate value:

rate index	avg. bitrate
0	7.7 kbps
1	9.8 kbps
2	14.3 kbps
3	20.8 kbps
4	27.9 kbps
5	34.2 kbps
6	(reserved)
7	NO_DATA

The value of 6 is reserved. If receiving this value, the packet MUST be discarded.

3.4. Speech Payload Table of Contents

The speech TOC is a bitmask indicating the presence of each frame in the packet. TOC is only available if the 'CR' value is not equal to 7 (NO_DATA).

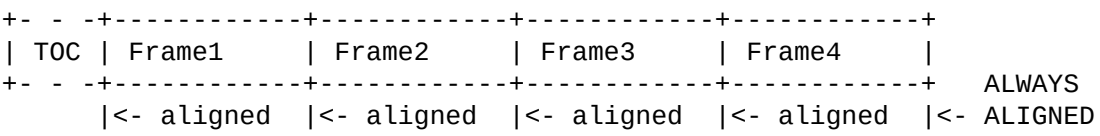


- E (1 bit): Frame existence indicator. The value of 0 indicates speech data is not present for the corresponding frame. The IP-MR encoder sets the 'E' flag to 0 for the periods of silence in DTX mode. Applications MUST set this bit to 0 if the frame is known to be damaged.

3.5. Speech Payload Data

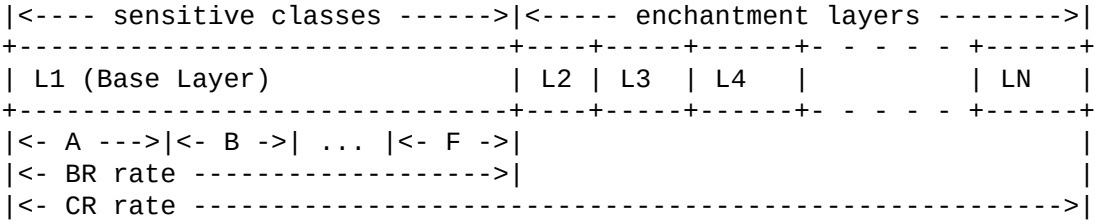
Speech data contains (GR+1) compressed IP-MR frames (20 ms of data). A compressed frame has a length of zero if the corresponding TOC flag is zero.

The beginning of each compressed frame is aligned if the 'A' bit is nonzero, while the end of the speech payload is always aligned to a byte (8-bit) boundary:



Marked regions MUST be padded only if the 'A' bit is set to '1'.

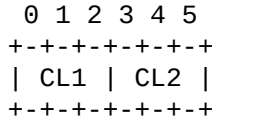
The compressed frame structure is as follows:



Appendix A of this document provides a helper routine written in "C" that MUST be used to extract sensitivity classes and bounds for the enchantment layers from the compressed frame data.

3.6. Redundancy Payload Header

The redundancy payload presence is signaled by the 'R' bit of the speech payload header. The redundancy header is composed of two fields of 3 bits each:



The 'CL1' and 'CL2' fields both specify the sensitivity classes available for preceding and pre-preceding packets respectively.

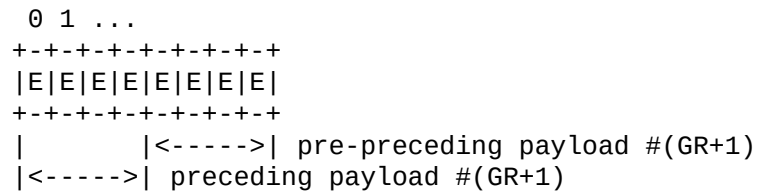
CL	Redundancy classes available
0	NONE
1	A
2	A-B
3	A-C
4	A-D
5	A-E
6	A-F
7	(reserved)

A receiver can reconstruct the base layer of preceding packets completely (CL=6) or partially (0<CL< 6) based on the sensitivity classes delivered. A decoder MUST discard the redundancy payload if 'CL' is equal to 0 or 7.

Note that the index of the base rate and grouping parameter is not transmitted for the redundancy payload. Applications MUST assume that 'BR' and 'GR' are the same as for the current packet.

3.7. Redundancy Payload Table of Contents

The redundancy TOC is a bitmask indicating the presence of each frame in the redundancy payload. The redundancy TOC is only available if the 'CL' value is not equal to 0 or 7.

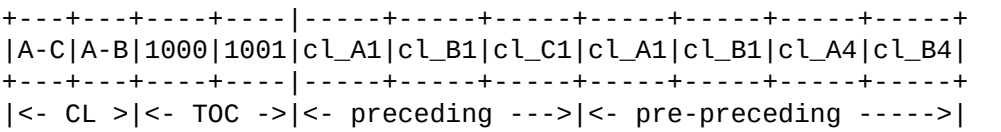


- o E (1 bit): Redundancy frame existence indicator. The value of 0 indicates redundancy data is not present for corresponding frame.

3.8. Redundancy Payload Data

IP-MR defines 6 classes ('A'-'F') of sensitivity to bit errors. Any damage of class 'A' bits causes significant reconstruction artifacts while the loss in class 'F' may not even be perceived by the listener. Note that only the base layer in a bitstream is represented as a set of classes. Together, the sensitivity classes' approach and redundancy allow IP-MR duplicate frames through the packets to improve robustness against packet loss.

Redundancy data carries a number of sensitivity classes for preceding and pre-preceding packets as indicated by the 'CL1' and 'CL2' fields of the redundancy header. The sensitivity classes' data is available individually for each frame only if the corresponding 'E' bit of the redundancy TOC is nonzero:



Redundancy data is only available if the base rates (BRs) and coding rates (CRs) of preceding and pre-preceding packets are the same as for the current packet.

A receiver MAY use redundancy data to compensate for packet loss (note that in this case, the 'CL' field MUST also be passed to the decoder). The helper routine provided in Appendix A MUST be used to extract sensitivity classes' length for each frame. The following pseudocode describes the sequence of operations:

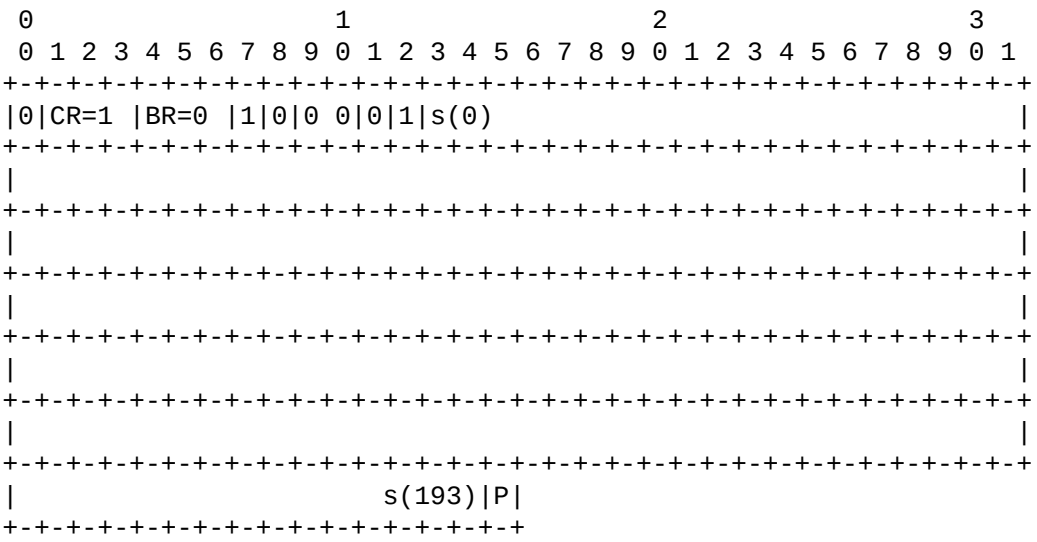
```
int sensitivityBits[numOfRedundancyFrames][6];
int redundancyBits [numOfRedundancyFrames];
for(i = 0 ; i < numOfRedundancyFrames; i++) {
    GetFrameInfo(CR, BR, pRedundancyPayloadData, dummy,
                 sensitivityBits[i], dummy);
    redundancyBits[i] = 0;
    for(j = 0; j < CL[i]; j++ ) {
         redundancyBits[i] += sensitivityBits[i][j];
    }
    flushBits(pRedundancyPayloadData, redundancyBits[i]);
}
```

4. Payload Examples

This section provides detailed examples of the IP-MR payload format.

4.1. Payload Carrying a Single Frame

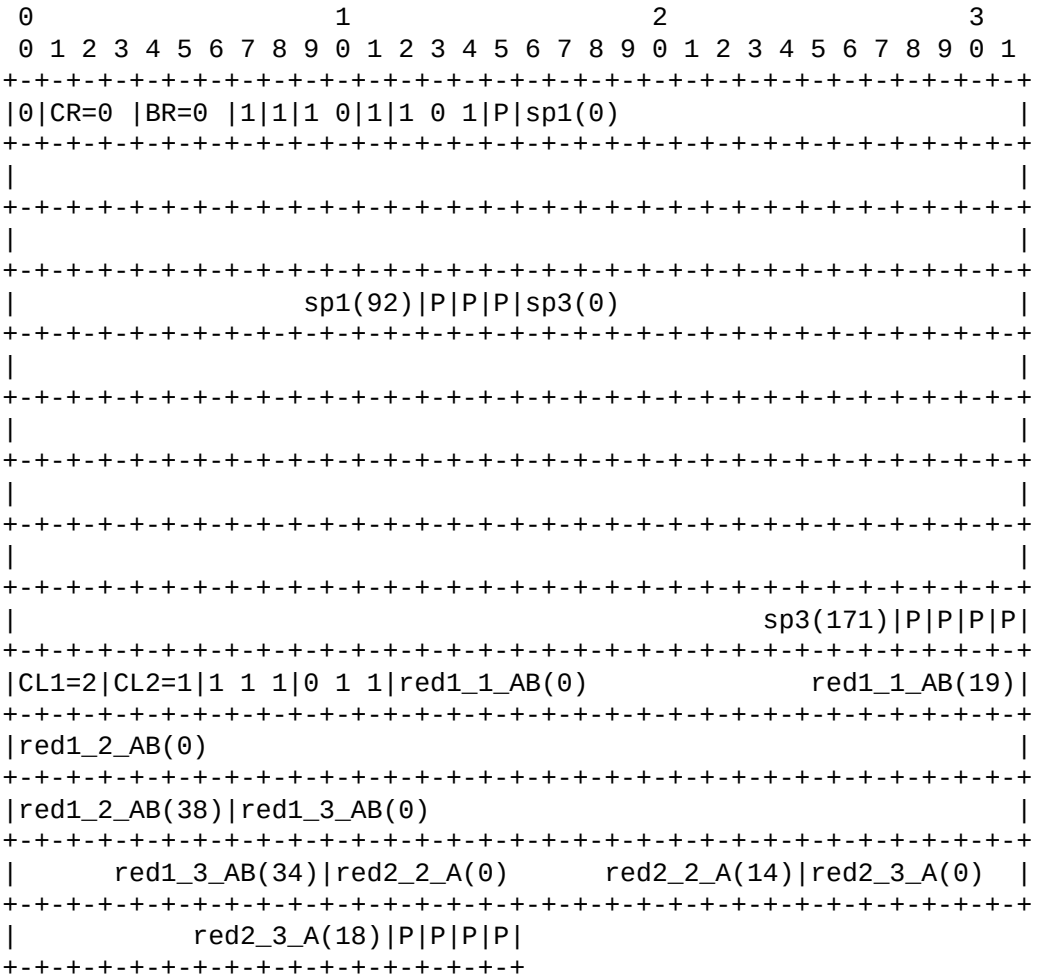
The following diagram shows a typical IP-MR payload carrying one (GR=0) non-aligned (A=0) speech frame without redundancy (R=0). The base layer is coded at 7.8 kbps (BR=0) while the coding rate is 9.7 kbps (CR=1). The 'E' bit value of 1 signals that compressed frame bits s(0) - s(193) are present. There is a padding bit 'P' to maintain speech payload size alignment.



4.2. Payload Carrying Multiple Frames with Redundancy

The following diagram shows a payload carrying 3 (GR=2) aligned (A=1) speech frames with redundancy (R=1). The TOC value of '101' indicates speech data present for the first (bits sp1(0)-sp1(92)) and third frames (bits sp3(0)-sp3(171)). There are no enchantment layers because the base and coding rates are equal (BR=CR=0). The padding bit 'P' is inserted to maintain necessary alignment.

The redundancy payload present for both preceding and pre-preceding payloads (CL1 = A-B, CL2=A), but redundancy data is only available for 5 (TOC='111011') of 6 (2*(GR+1)) frames. There is redundancy data of 20, 39, and 35 bits for each of the three frames of the preceding packet and 15 and 19 bits for the two frames of the pre-preceding packet.



5. Congestion Control

The general congestion control considerations for transporting RTP data applicable to IP-MR speech over RTP (see RTP [RFC3550] and any applicable RTP profile like the Audio-Visual Profile (AVP) [RFC3551]). However, the multi-rate capability of IP-MR speech coding provides a mechanism that may help to control congestion, since the bandwidth demand can be adjusted by selecting a different encoding mode.

The number of frames encapsulated in each RTP payload highly influences the overall bandwidth of the RTP stream due to header overhead constraints. Packetizing more frames in each RTP payload can reduce the number of packets sent and thus reduce the overhead from IP/UDP/RTP headers, at the expense of increased delay.

Due to the scalability nature of the IP_MR codec, the transmission rate can be reduced at any transport stage to fit channel bandwidth. The minimal rate is specified by the BR field of the payload header and can be as low as 7.7 kbps. It is up to the application to keep the balance between coding quality (high BR) and bitstream scalability (low BR). Because coding quality depends on coding rate (CR) rather than base rate (BR), it is NOT RECOMMENDED to use high BR values for real-time communications.

Applications MAY utilize bitstream redundancy to combat packet loss. However, the gateway is free to chose any option to reduce the transmission rate; the coding layer or redundancy bits can be dropped. Due to this fact, it is NOT RECOMMENDED for applications to increase the total bitrate when adding redundancy in response to packet loss.

6. Security Considerations

RTP packets using the payload format defined in this specification are subject to the security considerations discussed in the RTP specification [RFC3550] and in any applicable RTP profile. The main security considerations for the RTP packet carrying the RTP payload format defined within this memo are confidentiality, integrity, and source authenticity. Confidentiality is achieved by encryption of the RTP payload. Integrity of the RTP packets is achieved through a suitable cryptographic integrity-protection mechanism. Such a cryptographic system may also allow the authentication of the source of the payload. A suitable security mechanism for this RTP payload format should provide confidentiality, integrity protection, and source authentication at least capable of determining if an RTP packet is from a member of the RTP session.

Note that the appropriate mechanisms to provide security to RTP and payloads following this memo may vary. The security mechanisms are dependent on the application, the transport, and the signaling protocol employed. Therefore, a single mechanism is not sufficient; although if suitable, usage of the Secure Real-time Transport Protocol (SRTP) [RFC3711] is recommended. Other mechanisms that may be used are IPsec [RFC4301] and Transport Layer Security (TLS) [RFC5246] (RTP over TCP); other alternatives may exist.

This payload format does not exhibit any significant non-uniformity in the receiver-side computational complexity for packet processing and thus is unlikely to pose a denial-of-service threat due to the receipt of pathological data.

7. Payload Format Parameters

This section describes the media types and names associated with this payload format.

The IP-MR media subtype is defined as 'ip-mr_v2.5'. This subtype was registered to specify an internal codec version. Later, this version was accepted as final, the bitstream was frozen, and IP-MR v2.5 was published under the name of IP-MR. Currently, the terms 'IP-MR' and 'IP-MR v2.5' are synonyms. The subtype name 'ip-mr_v2.5' is being used in implementations.

7.1. Media Type Registration

Media Type name: audio

Media Subtype name: ip-mr_v2.5

Required parameters: none

Optional parameters:
These parameters apply to RTP transfer only.

ptime: The media packet length in milliseconds. Allowed values are: 20, 40, 60, and 80.

Encoding considerations:
This media type is framed and binary (see RFC 4288, Section 4.8).

Security considerations:
See Section 6 of RFC 6262.

Interoperability considerations:
none

Published specification:
RFC 6262

Applications that use this media type:
Real-time audio applications like voice over IP, teleconference, and multimedia streaming.

Additional information:
none

Person & email address to contact for further information:
V. Sviridenko <vladimirs@spiritdsp.com>

Intended usage:
COMMON

Restrictions on usage:
This media type depends on RTP framing and thus is only defined for transfer via RTP [RFC3550].

Authors:
Sergey Ikonin <info@spiritdsp.com>
Dmitry Yudin <info@spiritdsp.com>

Change controller:
IETF Audio/Video Transport working group delegated from the IESG.

7.2. Mapping Media Type Parameters into SDP

The information carried in the media type specification has a specific mapping to fields in the Session Description Protocol (SDP) [RFC4566], which is commonly used to describe RTP sessions. When SDP is used to specify sessions employing the IP-MR codec, the mapping is as follows:

- The media type ("audio") goes in SDP "m=" as the media name.
- The media subtype (payload format name) goes in SDP "a=rtpmap" as the encoding name. The RTP clock rate in "a=rtpmap" MUST be 16000.
- The parameter "ptime" goes in the SDP "a=ptime" attribute.

Any remaining parameters go in the SDP "a=fmtp" attribute by copying them directly from the media type parameter string as a semicolon-separated list of parameter=value pairs.

Note that the payload format (encoding) names are commonly shown in uppercase. Media subtypes are commonly shown in lowercase. These names are case-insensitive in both places.

8. IANA Considerations

One media type (ip-mr_v2.5) has been defined and registered in the media types registry.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

Appendix A. Retrieving Frame Information

This appendix contains the C code for implementation of the frame-parsing function. This function extracts information about a coded frame, including frame size, number of layers, size of each layer, and size of perceptual sensitive classes.

A.1. get_frame_info.c

```
/*

Copyright (c) 2011 IETF Trust and the persons identified as
authors of the code.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
  notice, this list of conditions and
  the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the
  distribution.

- Neither the name of Internet Society, IETF or IETF Trust, nor the
  names of specific contributors, may be used to endorse or promote
  products derived from this software without specific prior written
  permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*/

/*****************************************************************
```

```
  get_frame_info.c

  Retrieving frame information for IP-MR Speech Codec

*****************************************************************/

#define RATES_NUM       6   // number of codec rates
#define SENSE_CLASSES   6   // number of sensitivity classes (A..F)

// frame types
#define FT_SPEECH       0   // active speech
#define FT_DTX_SID      1   // silence insertion descriptor

// get specified bit from coded data
int GetBit(const unsigned char *buf, int curBit)
{
    return (buf[curBit>>3]>>(curBit%8))&1;
}

// retrieve frame information
int GetFrameInfo(               // o: frame size in bits
    short rate,                 // i: encoding rate (0..5)
    short base_rate,            // i: base (core) layer rate,
    const unsigned char buf[2], // i: coded bit frame
    int size,                   // i: coded bit frame size in bytes
    short pLayerBits[RATES_NUM],     // o: number of bits in layers
    short pSenseBits[SENSE_CLASSES], // o: number of bits in
                                     //    sensitivity classes
    short *nLayers                   // o: number of layers
)
{
    static const short Bits_1[4]    = {  0, 9, 9,15};
    static const short Bits_2[16]   = { 43,50,36,31,46,48,40,44,
                                        47,43,44,45,43,44,47,36};
    static const short Bits_3[2][6] = {{13,11,23,33,36,31},
                                       {25, 0,23,32,36,31},};
    int FrType;
    int i, nBits = 0;

    if (rate < 0 || rate > 5) {
        return 0; // incorrect stream
    }

    // extract frame type bit if required
    FrType = GetBit(buf, nBits++) ? FT_SPEECH : FT_DTX_SID;

    if((FrType != FT_DTX_SID && size < 2) || size < 1) {
        return 0; // not enough input data
```

```
    }

    for(i = 0; i < SENSE_CLASSES; i++) {
        pSenseBits[i] = 0;

    }

    {
        int cw_0;
        int b[14];

        // extract meaning bits
        for(i = 0 ; i < 14; i++) {
            b[i] = GetBit(buf, nBits++);
        }

        // parse
        if(FrType == FT_DTX_SID) {
            cw_0 = (b[0]<<0)|(b[1]<<1)|(b[2]<<2)|(b[3]<<3);
            rate = 0;
            pSenseBits[0] = 10 + Bits_2[cw_0];
        } else {

            int i, idx;
            int nFlag_1, nFlag_2, cw_1, cw_2;

            nFlag_1 = b[0] + b[2] + b[4] + b[6];
            cw_1 = (cw_1 << 1) | b[0];
            cw_1 = (cw_1 << 1) | b[2];
            cw_1 = (cw_1 << 1) | b[4];
            cw_1 = (cw_1 << 1) | b[6];

            nFlag_2 = b[1] + b[3] + b[5] + b[7];
            cw_2 = (cw_2 << 1) | b[1];
            cw_2 = (cw_2 << 1) | b[3];
            cw_2 = (cw_2 << 1) | b[5];
            cw_2 = (cw_2 << 1) | b[7];

            cw_0 = (b[10]<<0)|(b[11]<<1)|(b[12]<<2)|(b[13]<<3);
            if (base_rate < 0)    base_rate = 0;
            if (base_rate > rate) base_rate = rate;
            idx = base_rate == 0 ? 0 : 1;

            pSenseBits[0] = 15+Bits_2[cw_0];
            pSenseBits[1] = Bits_1[(cw_1>>0)&0x3] +
                            Bits_1[(cw_1>>2)&0x3];
            pSenseBits[2] = nFlag_1*5;
            pSenseBits[3] = nFlag_2*30;
```

```
                pSenseBits[5] = (4 - nFlag_2)*(Bits_3[idx][0]);

                for (i = 1; i < rate+1; i++) {
                    pLayerBits[i] = 4*Bits_3[idx][i];
                }

            }

            pLayerBits[0] = 0;
            for (i = 0; i < SENSE_CLASSES; i++) {
                pLayerBits[0] += pSenseBits[i];
            }

            *nLayers = rate+1;
        }

        {
            // count total frame size
            int payloadBitCount = 0;
            for (i = 0; i < *nLayers; i++) {
                payloadBitCount += pLayerBits[i];
            }
            return payloadBitCount;
        }
    }
```

Author's Address

Sergey Ikonin
SPIRIT DSP
Building 27, A. Solzhenitsyna Street
109004, Moscow
Russia

Tel: +7 495 661-2178
Fax: +7 495 912-6786
EMail: s.ikonin@gmail.com