Audio/Video Transport Working Group
Internet Draft
Intended status: Proposed Standard

S. Ikonin
SPIRIT DSP

October 13, 2010

RTP Payload Format for IP-MR Speech Codec
draft-ietf-avt-rtp-ipmr-14.txt

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 introduced redundancy for robustness against packet loss and bit errors.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

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/1id-abstracts.html

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

This Internet-Draft will expire on December 18, 2010.

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 introduced 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 [RFC 2119].

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. It means that not only source has the ability to change transmission rate on a 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.

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

In fact, not all of 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 cause significant reconstruction artifacts while the lost in class 'F' may be even not perceived by the listener. Note, only base layer in a bitstream is represented as a set of classes.

The IP-MR payload format allows frame duplicate through the packets to improve robustness against packet loss (Section 3.6). 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 application adjust the trade-off between overhead and robustness against packet loss. Note, this approach supported natively within a packet and requires no out-of-band signals or

session initialization procedures.

Main IP-MR features are as the following:

- 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 IP-MR payload.

3.1. RTP Header Usage

The format of the RTP header is specified in RFC 1889. 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, this corresponding 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 the talkspurt in Section 4.1 [RFC 3551]). 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.

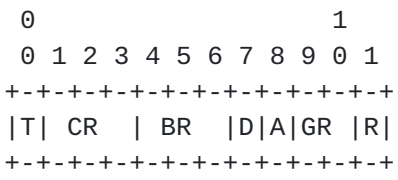
The remaining RTP header fields are used as specified in [RFC 3550].

3.2. RTP Payload Structure

The IP-MR payload composed of two payloads, one for current (speech) speech and one for redundancy. Both of payloads are represented in a form of: 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 which are common for all frames in the packet:



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

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

- o 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 as having a base rate of BR.

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

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

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

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

Note, the values of 'T' and 'D' bits are fixed, any other values are not allowed by specification. Note, the values of padding bit is not specified.

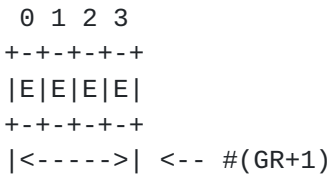
The following table defines mapping between rate index and rate value:

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

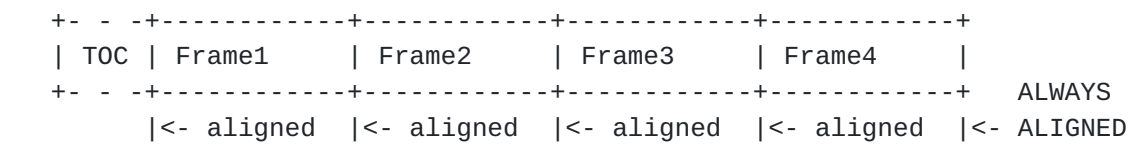


- o E (1 bit): Frame existence indicator. The value of 0 indicates speech data does not present for corresponding frame. IP-MR encoder sets E flag to 0 for the periods of silence in DTX mode. Application MUST set this bit to 0 if the frame is known to be damaged.

3.5. Speech Payload Data

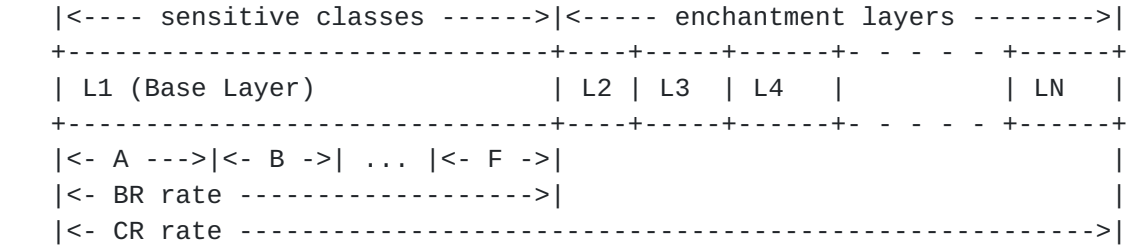
Speech data contains (GR+1) compressed IP-MR frames (20ms of data). Compressed frame have zero length if corresponding TOC flag is zero.

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



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

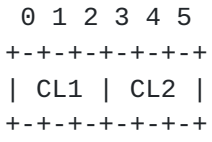
The compressed frame structure is the following:



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

3.6. Redundancy Payload Header

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



Both of 'CL1' and 'CL2' fields specify the sensitivity classes available for preceding and pre-preceding packets correspondingly.

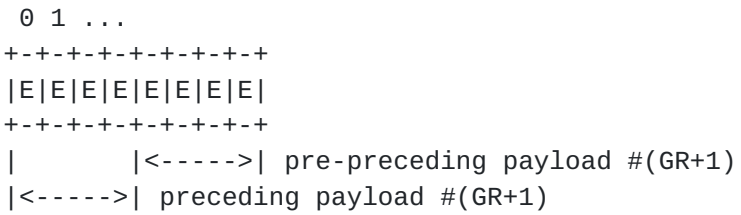
CL	Redundancy classes available
0	NONE
1	A
2	A-B
3	A-C
4	A-D
5	A-E
6	A-F
7	(reserved)

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

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

3.7. Redundancy Payload Table of Contents

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



o E (1 bit): Redundancy frame existence indicator. The value of 0 indicates redundancy data does 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 cause significant reconstruction artifacts while the lost in class 'F' may be even not perceived by the listener. Note, only base layer in a bitstream is represented as a set of classes. Together, the set of sensitivity classes approach and redundancy allows 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 'CL1' and 'CL2' fields of redundancy header. The sensitivity classes data is available individually for each frame only if corresponding 'E' bit of redundancy TOC is nonzero:

```
+---+---+----+----|-----+-----+-----+-----+-----+-----+-----+
|A-C|A-B|1000|1001|cl_A1|cl_B1|cl_C1|cl_A1|cl_B1|cl_A4|cl_B4|
+---+---+----+----|-----+-----+-----+-----+-----+-----+-----+
|<- CL >|<- TOC ->|<- preceding --->|<- pre-preceding ----->|
```

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

Receiver MAY use redundancy data to compensate packet loss, note this case the 'CL' field MUST be also passed to decoder. Helper routine provided in Annex A MUST be used to extract sensitivity classes length for each frame. The following pseudo code describes the sequence of operations:

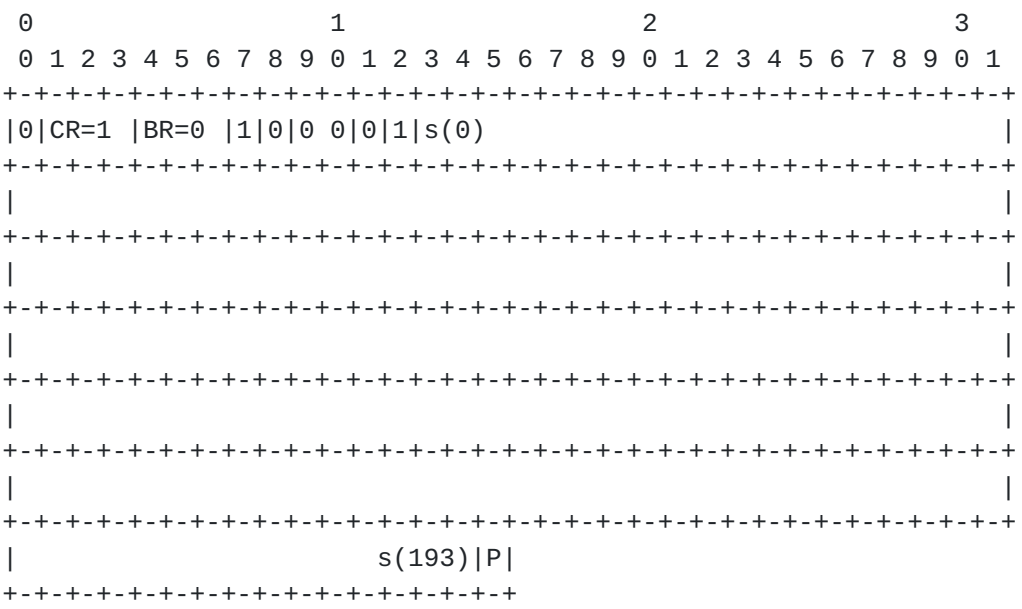
```
int sensitivityBits[numOfRedundancyFrames][6];
int redundancyBits [numOfRedundancyFrames];
for(i = 0 ; i < numOfRedundancyFrames; i++) {
    GetFrameInfo(CR, BR, pRedundancyPayloadData, dummy,
                 sensitivityBits[i], dummy);
    redundancyBits[i] = 0;
    for(j = 0; j < CL[i]; j++ )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

4. Payload Examples

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

4.1. Payload Carrying a Single Frame

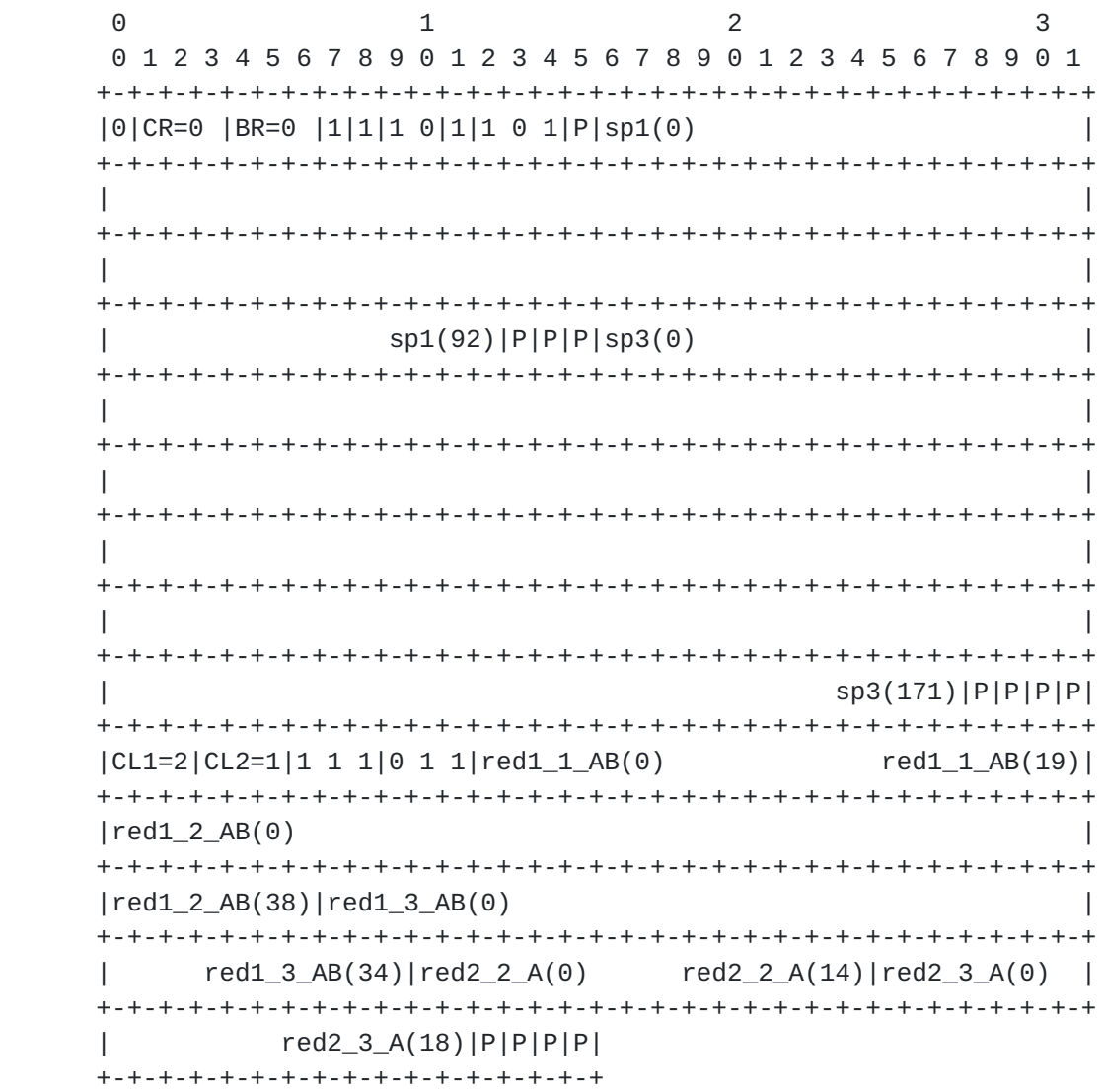
The following diagram shows typical IP-MR payload carrying a one (GR=0) non-aligned (A=0) speech frame without redundancy (R=0). The base layer is coded at 7.8 kbps (BR=0) while the coding rate is 9.7 kbps (CR=1). The 'E' bit value of 1 signals that compressed frame bits s(0) - s(193) are present. There 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 presents for a first (bits sp1(0)-sp1(92)) and third frames (bits sp3(0)-sp3(171)). There is no enchantment layers because of base and coding rates are equal (BR=CR=0). Padding bit 'P' is inserted to maintain necessary alignment.

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



5. Congestion Control The general congestion control considerations for transporting RTP data applicable to IP-MR speech over RTP (see RTP [RFC 3550] and any applicable RTP profile like AVP [RFC 3551]). 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 hence the overhead from IP/UDP/RTP headers, at the expense of increased delay.

Due to scalability nature of IP_MR codec the transmission rate can be reduced at any transport stage to fit channel bandwidth. The minimal rate is specified by BR field of payload header and can be is low as

7.7 kbps. It is up to application to keep balance between coding quality (high BR) and bitstream scalability (small BR). Because of coding quality depends rather on coding rate(CR) than base rate (BR), it is NOT RECOMMENDED to use high BR values for real-time communications.

Application MAY utilize bitstream redundancy to combat packet loss. But the gateway is free to chose any option to reduce transmission rate - coding layer or redundancy bits can be dropped. Due to this fact it is NOT RECOMMENDED application to increase total bitrate when adding redundancy in a 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 [RFC 3550] 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 at least source authentication capable of determining if an RTP packet is from a member of the RTP session.

Note that the appropriate mechanism to provide security to RTP and payloads following this memo may vary. It is 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) [RFC 3711] is recommended. Other mechanisms that may be used are IPsec [RFC 4301] and Transport Layer Security (TLS) [RFC 5246] (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 internal codec version. Later, this version was accepted as the final, the bitstream was frozen and IP-MR v2.5 was published under the name of IP-MR. Currently 'IP-MR' and 'IP-MR v2.5' terms are synonyms. The subtype name ip-mr_v2.5 is being uses 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 binary data (see RFC 4288, Section 4.8).

Security considerations:
See section 6 of RFC XXXX (RFC editor please replace with this RFC number).

Interoperability considerations:
none

Published specification:
RFC XXXX (RFC editor please replace with this RFC number)

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

Additional information:
none

Person & email address to contact for further information:
Dmitry Yudin <yudin@spiritdsp.com>

Intended usage:
COMMON

Restrictions on usage:
This media type depends on RTP framing, and hence is only defined

fortransfer via RTP [RFC 3550].

Authors:
Sergey Ikonin <info@spiritdsp.com> Dmitry Yudin
<yudin@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) [RFC 4566], 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 16000.
- The parameter "ptime" goes in the SDP "a=ptime" attributes.

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 upper case. Media subtypes are commonly shown in lower case. These names are case-insensitive in both places.

8. IANA Considerations

One media type has been defined and needs registration in the media types registry.

9. Normative References

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

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

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

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

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

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

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

10. Disclaimer

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

11. Legal Terms

All IETF Documents and the information contained therein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST 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 THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

The IETF Trust 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 any IETF 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.

Copies of Intellectual Property 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 any standard or specification contained in an IETF Document. Please address the information to the IETF at ietf-ipr@ietf.org.

The definitive version of an IETF Document is that published by, or under the auspices of, the IETF. Versions of IETF Documents that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of IETF Documents. The definitive version of these Legal Provisions is that published by, or under the auspices of, the IETF. Versions of these Legal Provisions that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of these Legal Provisions.

For the avoidance of doubt, each Contributor to the IETF Standards Process licenses each Contribution that he or she makes as part of the IETF Standards Process to the IETF Trust pursuant to the provisions of RFC 5378. No language to the contrary, or terms, conditions or rights that differ from or are inconsistent with the rights and licenses granted under RFC 5378, shall have any effect and shall be null and void, whether published or posted by such Contributor, or included with or in such Contribution.

12. Authors' Addresses

SPIRIT DSP
Building 27, A. Solzhenitsyna street
109004, Moscow, RUSSIA

Tel: +7 495 661-2178
Fax: +7 495 912-6786
EMail: yudin@spiritdsp.com

APPENDIX A. RETRIEVING FRAME INFORMATION

This appendix contains the c-code for implementation of frame parsing function. This function extracts information about 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) 2010
  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 BELIABLE 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(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,
       unsigned char *buf, // 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;
        }
    }
```