Internet Engineering Task Force (IETF)
Request for Comments: 6465
Category: Standards Track
ISSN: 2070-1721

E. Ivov, Ed.
Jitsi
E. Marocco, Ed.
Telecom Italia
J. Lennox
Vidyo

December 2011

A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication

Abstract

This document describes a mechanism for RTP-level mixers in audio conferences to deliver information about the audio level of individual participants. Such audio level indicators are transported in the same RTP packets as the audio data they pertain to.

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/rfc6465.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

"A Framework for Conferencing with the Session Initiation Protocol (SIP)" [RFC4353] presents an overall architecture for multi-party conferencing. Among others, the framework borrows from RTP [RFC3550] and extends the concept of a mixer entity "responsible for combining the media streams that make up a conference, and generating one or more output streams that are delivered to recipients". Every participant would hence receive, in a flat single stream, media originating from all the others.

Using such centralized mixer-based architectures simplifies support for conference calls on the client side, since they would hardly differ from one-to-one conversations. However, the method also introduces a few limitations. The flat nature of the streams that a mixer would output and send to participants makes it difficult for users to identify the original source of what they are hearing.

Mechanisms that allow the mixer to send to participants cues on current speakers (e.g., the contributing source (CSRC) fields in RTP [RFC3550]) only work for speaking/silent binary indications. There are, however, a number of use cases where one would require more detailed information. Possible examples include the presence of background chat/noise/music/typing, someone breathing noisily in their microphone, or other cases where identifying the source of the disturbance would make it easy to remove it (e.g., by sending a private IM to the concerned party asking them to mute their microphone). A more advanced scenario could involve an intense discussion between multiple participants that the user does not personally know. Audio level information would help better recognize the speakers by associating with them complex (but still human readable) characteristics like loudness and speed, for example.

One way of presenting such information in a user-friendly manner would be for a conferencing client to attach audio level indicators to the corresponding participant-related components in the user interface. One possible example is displayed in Figure 1, where levels can help users determine that Alice is currently the active speaker, Carol is mute, and Bob and Dave are sending some background noise.

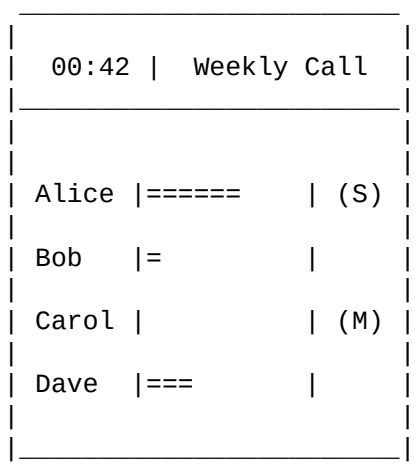

Figure 1: Displaying Detailed Speaker Information to the User by Including Audio Level for Every Participant

Implementing a user interface like the above requires analysis of the media sent from other participants. In a conventional audio conference, this is only possible for the mixer, since all other conference participants are generally receiving a single, flat audio stream and therefore have no immediate way of determining individual audio levels.

This document specifies an RTP extension header that allows such mixers to deliver audio level information to conference participants by including it directly in the RTP packets transporting the corresponding audio data.

The header extension in this document is different than, but complementary to, the one defined in [RFC6464], which defines a mechanism by which clients can indicate to audio mixers the levels of the audio in the packets they send.

2. Terminology

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].

3. Protocol Operation

According to RFC 3550 [RFC3550], a mixer is expected to include in outgoing RTP packets a list of identifiers (CSRC IDs) indicating the sources that contributed to the resulting stream. The presence of such CSRC IDs allows RTP clients to determine, in a binary way, the active speaker(s) in any given moment. The RTP Control Protocol (RTCP) also provides a basic mechanism to map the CSRC IDs to user identities through the CNAME field. More advanced mechanisms can exist, depending on the signaling protocol used to establish and control a conference. In the case of the Session Initiation Protocol [RFC3261], for example, "A Session Initiation Protocol (SIP) Event Package for Conference State" [RFC4575] defines a <src-id> tag that binds CSRC IDs to media streams and SIP URIs.

This document describes an RTP header extension that allows mixers to indicate the audio level of every contributing conference participant (CSRC) in addition to simply indicating their on/off status. This new header extension uses the general mechanism for RTP header extensions as described in [RFC5285].

Each instance of this header contains a list of one-octet audio levels expressed in -dBov, with values from 0 to 127 representing 0 to -127 dBov (see Figures 2 and 3). Appendix A provides a reference implementation indicating one way of obtaining such values from raw audio samples.

Every audio level value pertains to the CSRC identifier located at the corresponding position in the CSRC list. In other words, the first value would indicate the audio level of the conference participant represented by the first CSRC identifier in that packet, and so forth. The number and order of these values MUST therefore match the number and order of the CSRC IDs present in the same packet.

When encoding audio level information, a mixer SHOULD include in a packet information that corresponds to the audio data being transported in that same packet. It is important that these values follow the actual stream as closely as possible. Therefore, a mixer SHOULD also calculate the values after the original contributing stream has undergone possible processing such as level normalization, and noise reduction, for example.

It can sometimes happen that a conference involves more than a single mixer. In such cases, each of the mixers MAY choose to relay the CSRC list and audio level information they receive from peer mixers (as long as the total CSRC count remains below 16). Given that the maximum audio level is not precisely defined by this specification, it is likely that in such situations average audio levels would be perceptibly different for the participants located behind the different mixers.

4. Audio Levels

The audio level header extension carries the level of the audio in the RTP payload of the packet with which it is associated. This information is carried in an RTP header extension element as defined by "A General Mechanism for RTP Header Extensions" [RFC5285].

The payload of the audio level header extension element can be encoded using either the one-byte or two-byte header defined in [RFC5285]. Figures 2 and 3 show sample audio level encodings with each of these header formats.

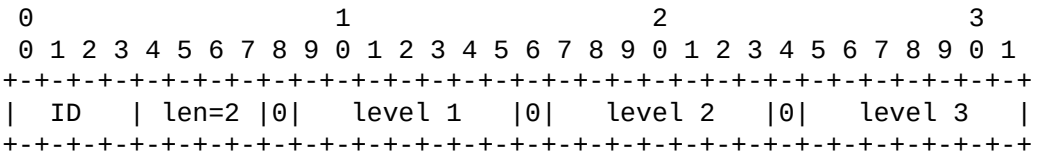

Figure 2: Sample Audio Level Encoding Using the One-Byte Header Format

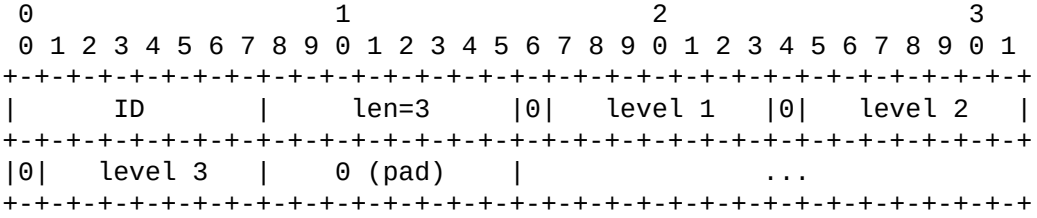

Figure 3: Sample Audio Level Encoding Using the Two-Byte Header Format

In the case of the one-byte header format, the 4-bit len field is the number minus one of data bytes (i.e., audio level values) transported in this header extension element following the one-byte header. Therefore, the value zero in this field indicates that one byte of data follows. In the case of the two-byte header format, the 8-bit len field contains the exact number of audio levels carried in the

extension. RFC 3550 [RFC3550] only allows RTP packets to carry a maximum of 15 CSRC IDs. Given that audio levels directly refer to CSRC IDs, implementations MUST NOT include more than 15 audio level values. The maximum value allowed in the len field is therefore 14 for the one-byte header format and 15 for the two-byte header format.

> Note: Audio levels in this document are defined in the same manner as is audio noise level in the RTP Payload Comfort Noise specification [RFC3389]. In [RFC3389], the overall magnitude of the noise level in comfort noise is encoded into the first byte of the payload, with spectral information about the noise in subsequent bytes. This specification's audio level parameter is defined so as to be identical to the comfort noise payload's noise-level byte.

The magnitude of the audio level itself is packed into the seven least significant bits of the single byte of the header extension, shown in Figures 2 and 3. The least significant bit of the audio level magnitude is packed into the least significant bit of the byte. The most significant bit of the byte is unused and always set to 0.

The audio level is expressed in -dBov, with values from 0 to 127 representing 0 to -127 dBov. dBov is the level, in decibels, relative to the overload point of the system, i.e., the highest-intensity signal encodable by the payload format. (Note: Representation relative to the overload point of a system is particularly useful for digital implementations, since one does not need to know the relative calibration of the analog circuitry.) For example, in the case of u-law (audio/pcmu) audio [ITU.G711], the 0 dBov reference would be a square wave with values +/- 8031. (This translates to 6.18 dBm0, relative to u-law's dBm0 definition in Table 6 of [ITU.G711].)

The audio level for digital silence -- for a muted audio source, for example -- MUST be represented as 127 (-127 dBov), regardless of the dynamic range of the encoded audio format.

The audio level header extension only carries the level of the audio in the RTP payload of the packet with which it is associated, with no long-term averaging or smoothing applied. That level is measured as a root mean square of all the samples in the measured range.

To simplify implementation of the encoding procedures described here, this specification provides a sample Java implementation (see Appendix A) of an audio level calculator that helps obtain such values from raw linear Pulse Code Modulation (PCM) audio samples.

5. Signaling Information

The URI for declaring the audio level header extension in a Session Description Protocol (SDP) extmap attribute and mapping it to a local extension header identifier is "urn:ietf:params:rtp-hdrext:csrc-audio-level". There is no additional setup information needed for this extension (i.e., no extension attributes).

An example attribute line in the SDP for a conference might be:

```
a=extmap:7 urn:ietf:params:rtp-hdrext:csrc-audio-level
```

The above mapping will most often be provided per media stream (in the media-level section(s) of SDP, i.e., after an "m=" line) or globally if there is more than one stream containing audio level indicators in a session.

Presence of the above attribute in the SDP description of a media stream indicates that RTP packets in that stream, which contain the level extension defined in this document, will be carrying such an extension with an ID of 7.

Conferencing clients that support audio level indicators and have no mixing capabilities would not be able to provide content for this audio level extension and would hence have to always include the direction parameter in the "extmap" attribute with a value of "recvonly". Conference focus entities with mixing capabilities can omit the direction or set it to "sendrecv" in SDP offers. Such entities would need to set it to "sendonly" in SDP answers to offers with a "recvonly" parameter and to "sendrecv" when answering other "sendrecv" offers.

This specification only defines the use of the audio level extensions in audio streams. They MUST NOT be advertised with other media types, such as video or text, for example.

Figures 4 and 5 show two example offer/answer exchanges between a conferencing client and a focus, and between two conference focus entities.

SDP Offer:

```
v=0
o=alice 2890844526 2890844526 IN IP6 host.example.com
s=-
c=IN IP6 host.example.com
t=0 0
m=audio 49170 RTP/AVP 0 4
a=rtpmap:0 PCMU/8000
a=rtpmap:4 G723/8000
a=extmap:1/recvonly urn:ietf:params:rtp-hdrext:csrc-audio-level
```

SDP Answer:

```
v=0
i=A Seminar on the session description protocol
o=conf-focus 2890844730 2890844730 IN IP6 focus.example.net
s=-
c=IN IP6 focus.example.net
t=0 0
m=audio 52544 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=extmap:1/sendonly urn:ietf:params:rtp-hdrext:csrc-audio-level
```

Figure 4: A Client-Initiated Example SDP Offer/Answer Exchange Negotiating an Audio Stream with One-Way Flow of Audio Level Information

SDP Offer:

```
v=0
i=Un seminaire sur le protocole de description des sessions
o=fr-focus 2890844730 2890844730 IN IP6 focus.fr.example.net
s=-
c=IN IP6 focus.fr.example.net
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=extmap:1/sendrecv urn:ietf:params:rtp-hdrext:csrc-audio-level
```

SDP Answer:

```
v=0
i=A Seminar on the session description protocol
o=us-focus 2890844526 2890844526 IN IP6 focus.us.example.net
s=-
c=IN IP6 focus.us.example.net
t=0 0
m=audio 52544 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=extmap:1/sendrecv urn:ietf:params:rtp-hdrext:csrc-audio-level
```

Figure 5: An Example SDP Offer/Answer Exchange between Two Conference Focus Entities with Mixing Capabilities Negotiating an Audio Stream with Bidirectional Flow of Audio Level Information

6. Security Considerations

1. This document defines a means of attributing audio level to a particular participant in a conference. An attacker may try to modify the content of RTP packets in a way that would make audio activity from one participant appear to be coming from another participant.

2. Furthermore, the fact that audio level values would not be protected even in a Secure Real-time Transport Protocol (SRTP) session [RFC3711] might be of concern in some cases where the activity of a particular participant in a conference is confidential. Also, as discussed in [SRTP-VBR-AUDIO], an attacker might be able to infer information about the conversation, possibly with phoneme-level resolution.

3. Both of the above are concerns that stem from the design of the RTP protocol itself, and they would probably also apply when using CSRC identifiers in the way specified in RFC 3550 [RFC3550]. It is therefore important that, according to the

needs of a particular scenario, implementors and deployers consider the use of header extension encryption [SRTP-ENCR-HDR] or a lower-level security and authentication mechanism such as IPsec [RFC4301], for example.

7. IANA Considerations

This document defines a new extension URI in the RTP Compact Header Extensions subregistry of the Real-Time Transport Protocol (RTP) Parameters registry, according to the following data:

```
Extension URI: urn:ietf:params:rtp-hdrext:csrc-audio-level
Description:   Mixer-to-client audio level indicators
Contact:       emcho@jitsi.org
Reference:     RFC 6465
```

8. Acknowledgments

Lyubomir Marinov contributed level measurement and rendering code.

Keith Drage, Roni Even, Miguel A. Garcia, John Elwell, Kevin P. Fleming, Ingemar Johansson, Michael Ramalho, Magnus Westerlund, and several others provided helpful feedback over the avt and avtext mailing lists.

Jitsi's participation in this specification is funded by the NLnet Foundation.

9. References

9.1. 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.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

9.2. Informative References

[ITU.G711] International Telecommunication Union, "Pulse Code Modulation (PCM) of Voice Frequencies", ITU-T Recommendation G.711, November 1988.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, September 2002.

[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.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, Ed., "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6465, December 2011.

[SRTP-ENCR-HDR] Lennox, J., "Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)", Work in Progress, October 2011.

[SRTP-VBR-AUDIO] Perkins, C. and JM. Valin, "Guidelines for the use of Variable Bit Rate Audio with Secure RTP", Work in Progress, July 2011.

Appendix A. Reference Implementation

This appendix contains Java code for a reference implementation of the level calculation and rendering methods. The code is not normative and is by no means the only possible implementation. Its purpose is to help implementors add audio level support to mixers and clients.

The Java code contains an AudioLevelCalculator class that calculates the sound pressure level of a signal with specific samples. It can be used in mixers to generate values suitable for the level extension headers.

The implementation is provided in Java but does not rely on any of the language specifics and can be easily ported to another language.

A.1. AudioLevelCalculator.java

<CODE BEGINS>

```
/*
   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, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents (http://trustee.ietf.org/license-info).
*/

/**
 * Calculates the audio level of specific samples of a signal
 * relative to overload.
 */
public class AudioLevelCalculator
{

    /**
     * Calculates the audio level of a signal with specific
     * <tt>samples</tt>.
     *
     * @param samples  the samples whose audio level we need to
     * calculate.  The samples are specified as an <tt>int</tt>
     * array starting at <tt>offset</tt>, extending <tt>length</tt>
     * number of elements, and each <tt>int</tt> element in the
     * specified range representing a sample whose audio level we
```

```
         * need to calculate.  Though a sample is provided in the
         * form of an <tt>int</tt> value, the sample size in bits
         * is determined by the caller via <tt>overload</tt>.
         *
         * @param offset  the offset in <tt>samples</tt> at which the
         * samples start.
         *
         * @param length  the length of the signal specified in
         * <tt>samples<tt>, starting at <tt>offset</tt>.
         *
         * @param overload  the overload (point) of <tt>signal</tt>.
         * For example, <tt>overload</tt> can be {@link Byte#MAX_VALUE}
         * for 8-bit signed samples or {@link Short#MAX_VALUE} for
         * 16-bit signed samples.
         *
         * @return  the audio level of the specified signal.
         */
        public static int calculateAudioLevel(
            int[] samples, int offset, int length,
            int overload)
        {
            /*
             * Calculate the root mean square (RMS) of the signal.
             */
            double rms = 0;

            for (; offset < length; offset++)
            {
                double sample = samples[offset];

                sample /= overload;
                rms += sample * sample;
            }
            rms = (length == 0) ? 0 : Math.sqrt(rms / length);

            /*
             * The audio level is a logarithmic measure of the
             * rms level of an audio sample relative to a reference
             * value and is measured in decibels.
             */
            double db;

            /*
             * The minimum audio level permitted.
             */
            final double MIN_AUDIO_LEVEL = -127;
```

```
            /*
             * The maximum audio level permitted.
             */
            final double MAX_AUDIO_LEVEL = 0;

            if (rms > 0)
            {
                /*
                 * The "zero" reference level is the overload level,
                 * which corresponds to 1.0 in this calculation, because
                 * the samples are normalized in calculating the RMS.
                 */
                db = 20 * Math.log10(rms);

                /*
                 * Ensure that the calculated level is within the minimum
                 * and maximum range permitted.
                 */
                if (db < MIN_AUDIO_LEVEL)
                    db = MIN_AUDIO_LEVEL;
                else if (db > MAX_AUDIO_LEVEL)
                    db = MAX_AUDIO_LEVEL;
            }
            else
            {
                db = MIN_AUDIO_LEVEL;
            }

            return (int)Math.round(db);
        }
    }

    <CODE ENDS>
```

Authors' Addresses

Emil Ivov (editor)
Jitsi
Strasbourg 67000
France

EMail: emcho@jitsi.org

Enrico Marocco (editor)
Telecom Italia
Via G. Reiss Romoli, 274
Turin 10148
Italy

EMail: enrico.marocco@telecomitalia.it

Jonathan Lennox
Vidyo, Inc.
433 Hackensack Avenue
Seventh Floor
Hackensack, NJ 07601
US

EMail: jonathan@vidyo.com