Network Working Group M. Petit-Huguenin
Internet-Draft Unaffiliated
Updates: 3550 (if approved) G. Zorn, Ed.
Intended status: Standards Track Network Zen
Expires: October 25, 2012 April 23, 2012

Support for Multiple Clock Rates in an RTP Session
draft-ietf-avtext-multiple-clock-rates-04

Abstract

This document clarifies the RTP specification when different clock rates are used in an RTP session. It also provides guidance on how to interoperate with legacy RTP implementations that use multiple clock rates.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The clock rate is a parameter of the payload format. It is often defined as been the same as the sampling rate but it is not always the case (see e.g. the G722 and MPA audio codecs [RFC3551]).

An RTP sender can switch between different payloads during the lifetime of an RTP session and because clock rates are defined by payload types, it is possible that the clock rate also varies during an RTP session. Schulzrinne, et al. [RFC3550] lists using multiple clock rates as one of the reasons to not use different payloads on the same SSRC but unfortunately this advice was not always followed and some RTP implementations change the payload in the same SSRC even if the different payloads use different clock rates.

This creates three problems:

- The method used to calculate the RTP timestamp field in an RTP packet is underspecified.

- When the same SSRC is used for different clock rates, it is difficult to know what clock rate was used for the RTP timestamp field in an RTCP SR packet.

- When the same SSRC is used for different clock rates, it is difficult to know what clock rate was used for the interarrival jitter field in an RTCP RR packet.

Table 1 contains a non-exhaustive list of fields in RTCP packets that uses a clock rate as unit:

Field name	RTCP packet type	Reference
RTP timestamp	SR	[RFC3550]
Interarrival jitter	RR	[RFC3550]
min_jitter	XR Summary Block	[RFC3611]
max_jitter	XR Summary Block	[RFC3611]
mean_jitter	XR Summary Block	[RFC3611]
dev_jitter	XR Summary Block	[RFC3611]
Interarrival jitter	IJ	[RFC5450]
RTP timestamp	SMPTETC	[RFC5484]
Jitter	RSI Jitter Block	[RFC5760]
Median jitter	RSI Stats Block	[RFC5760]

Table 1

This document first tries to list in Section 3 and subsections all of the algorithms known to be used in existing RTP implementations at the time of writing. These sections are not normative.

Section 4 and subsections then recommend a unique algorithm that modifies RFC 3550. These sections are normative.

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]. In addition, this document uses the following terms:

Clock rate
: The multiplier used to convert from a wallclock value in seconds to an equivalent RTP timestamp value (without the fixed random offset). Note that RFC 3550 uses various terms like "clock frequency", "media clock rate", "timestamp unit", "timestamp frequency", and "RTP timestamp clock rate" as synonymous to clock rate.

RTP Sender
: A logical network element that sends RTP packets, sends RTCP SR packets, and receives RTCP RR packets.

RTP Receiver
: A logical network element that receives RTP packets, receives RTCP SR packets, and sends RTCP RR packets.

3. Legacy RTP

The following sections describe the various ways legacy RTP implementations behave when multiple clock rates are used. Legacy RTP refers to RFC 3550 without the modifications introduced by this document.

3.1. Different SSRC

One way of managing multiple clock rates is to use a different SSRC for each different clock rate, as in this case there is no ambiguity on the clock rate used by fields in the RTCP packets. This method also seems to be the original intent of RTP as can be deduced from points 2 and 3 of section 5.2 of RFC 3550.

On the other hand changing the SSRC can be a problem for some implementations designed to work only with unicast IP addresses, where having multiple SSRCs is considered a corner case. Lip

synchronization can also be a problem in the interval between the beginning of the new stream and the first RTCP SR packet. This is not different than what happen at the beginning of the RTP session but it can be more annoying for the end-user.

3.2. Same SSRC

The simplest way of managing multiple clock rates is to use the same SSRC for all the payload types regardless of the clock rates.

Unfortunately there is no clear definition on how the RTP timestamp should be calculated in this case. The following subsections present the algorithms used in the field.

3.2.1. Monotonic timestamps

This method of calculating the RTP timestamp ensures that the value increases monotonically. The formula used by this method is as follows:

```
timestamp = previous_timestamp
            + (current_capture_time - previous_capture_time)
            * current_clock_rate
```

The problem with this method is that the jitter calculation on the receiving side gives an invalid result during the transition between two clock rates, as shown in Table 2. The capture and arrival time are in seconds, starting at the beginning of the capture of the first packet; clock rate is in Hz; the RTP timestamp does not include the random offset; the transit, jitter, and average jitter use the clock rate as unit.

Capt. time	Clock rate	RTP timestamp	Arrival time	Transit	Jitter	Average jitter
0	8000	0	0.1	800		
0.02	8000	160	0.12	800	0	0
0.04	8000	320	0.14	800	0	0
0.06	8000	480	0.16	800	0	0
0.08	16000	800	0.18	2080	480	30
0.1	16000	1120	0.2	2080	0	28
0.12	16000	1440	0.22	2080	0	26
0.14	8000	1600	0.24	320	720	70
0.16	8000	1760	0.26	320	0	65

Table 2

Calculating the correct transit time on the receiving side can be done by using the following formulas:

1. current_time_capture = current_timestamp - previous_timestamp) / current_clock_rate + previous_time_capture

2. transit = current_clock_rate * (time_arrival - current_time_capture)

3. previous_time_capture = current_time_capture

The main problem with this method, in addition to the fact that the jitter calculation described in RFC 3550 cannot be used, is that is it dependent on the previous RTP packets, packets that can be reordered or lost in the network.

3.2.2. Non-monotonic timestamps

An alternate way of generating the RTP timestamps is to use the following formula:

timestamp = capture_time * clock_rate

With this formula, the jitter calculation is correct but the RTP timestamp values are no longer increasing monotonically as shown in Table 3. RFC 3550 states that "[t]he sampling instant MUST be derived from a clock that increments monotonically[...]" but nowhere says that the RTP timestamp must increment monotonically.

Capt. time	Clock rate	RTP timestamp	Arrival time	Transit	Jitter	Average jitter
0	8000	0	0.1	800		
0.02	8000	160	0.12	800	0	0
0.04	8000	320	0.14	800	0	0
0.06	8000	480	0.16	800	0	0
0.08	16000	1280	0.18	1600	0	0
0.1	16000	1600	0.2	1600	0	0
0.12	16000	1920	0.22	1600	0	0
0.14	8000	1120	0.24	800	0	0
0.16	8000	1280	0.26	800	0	0

Table 3

The advantage with this method is that it works with the jitter calculation described in RFC 3550, as long as the correct clock rates

are used. It seems that this is what most implementations are using.

4. Recommendations

4.1. RTP Sender

An RTP Sender with RTCP turned off (i.e. by setting the RS and RR bandwidth modifiers [RFC3556] to 0) SHOULD use a different SSRC for each different clock rate but MAY use different clock rates on the same SSRC as long as the RTP timestamp without the random offset is calculated as explained below:

[[This was designed to help VoIP implementations who anyway never cared about RTCP. Do we want to keep this?]]

Each time the clock rate changes, the start_offset and capture_start values are calculated with the following formulas:

```
start_offset = (capture_time - capture_start) * previous_clock_rate
capture_start = capture_time
```

For the first RTP packet, the values are initialized with the following values:

```
start_offset = 0
capture_start = capture_time
```

After eventually updating these values, the RTP timestamp is calculated with the following formula:

```
timestamp = (capture_time - capture_start) * clock_rate +
start_offset
```

Note that in all the formulas, capture_time is the first instant the new timestamp rate is used.

An RTP Sender with RTCP turned on MUST use a different SSRC for each different clock rate. An RTCP BYE MUST be sent and a new SSRC MUST be used if the clock rate switches back to a value already seen in the RTP stream.

To accelerate lip synchronization, the next compound RTCP packet sent by the RTP sender MUST contain multiple SR packets, the first one containing the mapping for the current clock rate and the next SR packets containing the mapping for the other clock rates seen during the last period.

[[Some legacy implementations may dislike receiving multiple SR packets. What should we do?]]

The RTP extension defined in Perkins & Schierl [RFC6051] MAY be used to accelerate the synchronization.

4.2. RTP Receiver

An RTP Receiver MUST calculate the jitter using the following formula:

D(i,j) = (arrival_time_j * clock_rate_i - timestamp_j) - (arrival_time_i * clock_rate_i - timestamp_i)

An RTP Receiver MUST be able to handle a compound RTCP packet with multiple SR packets.

For interoperability with legacy RTP implementations, an RTP receiver MAY use the information in two consecutive SR packets to calculate the clock rate used, i.e. if Ni is the NTP timestamp for the SR packet i, Ri the RTP timestamp for the SR packet i and Nj and Rj the NTP timestamp and RTP timestamp for the previous SR packet j, then the clock rate can be guessed as the closest to (Ri - Rj) / (Ni - Nj).

5. Security Considerations

TBD

6. IANA Considerations

This document requires no IANA actions.

7. Acknowledgements

Thanks to Colin Perkins, Ali C. Begen and Magnus Westerlund for their comments, suggestions and questions that helped to improve this document.

Thanks to Robert Sparks and the attendees of SIPit 26 for the survey on multiple clock rates interoperability.

This document was written with the xml2rfc tool described in Rose [RFC2629].

8. References

8.1. 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.

8.2. Informative References

[I-D.ietf-avt-variable-rate-audio]
Wenger, S. and C. Perkins, "RTP Timestamp Frequency for Variable Rate Audio Codecs", draft-ietf-avt-variable-rate-audio-00 (work in progress), October 2004.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

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

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.

[RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.

[RFC5450] Singer, D. and H. Desineni, "Transmission Time Offsets in RTP Streams", RFC 5450, March 2009.

[RFC5484] Singer, D., "Associating Time-Codes with RTP Streams", RFC 5484, March 2009.

[RFC5760] Ott, J., Chesterfield, J., and E. Schooler, "RTP Control Protocol (RTCP) Extensions for Single-Source Multicast Sessions with Unicast Feedback", RFC 5760, February 2010.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

Appendix A. Using a fixed clock rate

An alternate way of fixing the multiple clock rates issue was proposed in [I-D.ietf-avt-variable-rate-audio]. This document proposed to define a unified clock rate, but the proposal was rejected at IETF 61.

Appendix B. Behavior of Legacy Implementations

B.1. libccrtp 2.0.2

This library uses the formula described in Section 3.2.2.

Note that this library uses gettimeofday(2) which is not guaranteed to increment monotonically, like when the clock is adjusted by NTP.

B.2. libmediastreamer0 2.6.0

This library (which uses the oRTP library) uses the formula described in Section 3.2.2.

Note that in some environments this library uses gettimeofday(2) which is not guaranteed to increment monotonically.

B.3. libpjmedia 1.0

This library uses the formula described in Section 3.2.2.

B.4. Android RTP stack 4.0.3

This library changes the SSRC each time the format changes, as described in Section 3.1.

Authors' Addresses

Marc Petit-Huguenin
Unaffiliated

Email: petithug@acm.org

Glen Zorn (editor)
Network Zen
227/358 Thanon Sanphawut
Bang Na, Bangkok 10260
Thailand

Phone: +66 (0) 87-0404617
Email: glenzorn@gmail.com