Voice in Packets: RTP, RTCP, Header Compression, Playout Algorithms, Terminal Requirements and Implementations

Jani Lakkakorpi
Nokia Research Center
P.O. Box 407
FIN-00045 NOKIA GROUP
Finland
jani.lakkakorpi@nokia.com

Abstract
RTP/RTCP protocol suite provides the means for sending packetized voice by introducing time stamps and sequence numbers in packet headers. Playout buffering is needed to re-synchronize the received voice stream. In this paper, a new adaptive playout delay adjustment algorithm is introduced.

A major problem, especially on low-bandwidth links, with Voice over IP (VoIP) packets is that they include a lot of overhead. The solution is header compression, which is done on link-by-link basis.

All terminals that support real time interactive voice should have enough processing power. The computational requirements of voice codecs usually increase with the voice compression ratio.

1 Introduction
During an average telephone conversation, each party usually talks only 35 per cent of the time. Most of the techniques that are used today to transform voice into data have the ability to detect silent periods. This allows asynchronous voice transmission and statistical multiplexing [Her00].

With statistical multiplexing, bandwidth can be used more efficiently. However, statistical multiplexing also introduces some uncertainty in the network. This uncertainty is variation in delay – also known as jitter. It needs to be corrected by the receiving side by adding some playout delay in order to restore the original packet spacing. Otherwise the original speech would sound incomprehensible.

Real-Time Transport Protocol (RTP) gives us the means to re-synchronize a voice stream. Each RTP packet is equipped with a timestamp and a sequence number. Chapter 2 describes RTP and its companion protocol, Real-Time Control Protocol (RTCP). Chapter 3 addresses the problem of RTP header compression – the overhead can be really large, as we will see. Chapter 4 introduces some playout algorithms developed for packetized voice, and chapter 5 gives a short overview about the terminal requirements and implementations for Voice over IP.

2 RTP and RTCP
Real-Time Transport Protocol (RTP) provides end-to-end transport functions suitable for applications that transmit real time data, such as audio or video over multicast or unicast networks. RTP does not provide any Quality of Service (QoS) guarantees but it is only responsible of synchronizing the received packets – within a single stream or across two streams. To achieve this, RTP packets are equipped with timestamps and sequence numbers.

RTP is designed to work together with Real-Time Control Protocol (RTCP) to get feedback on the quality of data transmission and information about participants in the on-going session. Request For Comments (RFC) 1889 (and its later revisions) includes complete descriptions of these protocols and their uses [Sch96a].

2.1 RTP
The first twelve octets in an RTP header (the first three rows in Figure 1) are included in all RTP packets, while the list of Contributing Source identifiers (CSRC) is present only when inserted by a mixer (see Section 2.2).

- **Version (V, 2 bits)**
  This field identifies the RTP version. The current version, defined in RFC 1889, is two.

- **Padding (P, 1 bit)**
  If padding bit is set, packet contains one or more padding octets at the end of the payload. The last
octet of the payload contains the number of padding octets.

<table>
<thead>
<tr>
<th>V</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Synchronization Source (SSRC) Identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Contributing Source (CSRC) Identifiers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Profile-specific Extensions</td>
</tr>
</tbody>
</table>

Figure 1. RTP Header Format

- **Extension (X, 1 bit)**
  
  If extension bit is set, the fixed RTP header and possible CSRCs are followed by extensions that use the format defined in RFC 1889.

- **CSRC count (CC, 4 bits)**
  
  CSRC count contains the number of Contributing Source identifiers that follow the fixed header. This number is usually zero [Her00].

- **Marker (M, 1 bit)**
  
  The interpretation of marker bit is defined by RTP profile. The marker bit is intended for marking significant events, such as frame boundaries, in the packet stream. H.225.0, for example, says that for audio codings supporting silence suppression, the marker bit must be set to one in the first packet of each talkspurt after a silence period [Her00].

- **Payload Type (PT, 7 bits)**
  
  This field identifies the format of the RTP payload, and determines its interpretation by the application. A profile specifies default static mapping of payload type codes to payload formats. An initial set of default mappings for audio and video is specified in RFC 1889 [Sch96b].

- **Sequence Number (16 bits)**
  
  Sequence number starts from a random value and it is incremented by one for each RTP packet sent. The sequence number is used by the receiver to detect packet losses and to restore packet sequence.

- **Timestamp (32 bits)**
  
  Timestamp reflects the sampling instant of the first payload octet. The clock frequency is defined for each payload type, and the clock is initialized with a random value [Her00].

- **SSRC (32 bits)**
  
  SSRC field identifies the synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session have the same SSRC identifier.

- **CSRC list (0 to 15 items, 32 bits each)**
  
  CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. Only 15 sources can be identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of the contributing sources.

2.2 Mixers and Translators

RTP-level relays called mixers re-synchronize incoming packets in order to reconstruct the packet spacing generated by the sender, mix these reconstructed streams into a single stream, possibly translate the encoding, and then forward the packet stream. These packets might be unicast to a single recipient or multicast to multiple recipients. RTP header includes the means for mixers to identify the sources that contributed to a mixed packet so that correct talker indication, for example, can be provided at the receiver.

Another type of RTP-level relay, translator, just takes a stream and passes it through. Translator can be used, for example, in a situation where the receiver is beyond a firewall.

2.3 RTCP

Real-Time Control Protocol (RTCP) is based on the periodic transmission of control packets to all participants in the session. The underlying protocol must provide multiplexing of data and control packets, for example, by using separate port numbers with UDP (User Datagram Protocol). RTP is usually assigned an even UDP port, and RTCP the next odd UDP port. RTCP performs three mandatory functions:

1. The primary function is to provide feedback on the quality of the data distribution. This function is performed through sender and receiver reports.
2. RTCP carries a persistent transport-level identifier for an RTP source, called Canonical Name (CNAME). Since the SSRC identifier may change,
all receivers require the CNAME to keep track of each participant.

3. Since the first two functions require that all participants in a session send RTCP packets, the RTCP packet rate must be controlled in order to scale up to a large number of participants. Each participant can independently observe the number of other participants and thus control its RTCP packet rate. The maximum rate at which a participant can send RTCP reports is one per five seconds.

It is recommended that translators and mixers combine individual RTCP packets from multiple sources into compound packets whenever feasible.

RFC 1889 defines several RTCP packet types to carry control information:

- **SR (Sender Report)** contains transmission and reception statistics for active senders
- **RR (Receiver Report)** contains reception statistics for participants that are not active senders
- **SDES (Source Description Items)** describes various parameters about the source, including the CNAME
- **BYE** packet is sent by a participant when leaving the session
- **APP**: application specific functions

Each RTCP packet begins with a fixed part, followed by structured elements that may be of variable length according to the packet type, but always end on a 32-bit boundary.

### Sender and Receiver Reports

RTP receivers provide reception quality feedback using RTCP report packets of two types. The only difference between sender reports (SR) and receiver reports (RR), besides the packet type code, is that the sender report includes a 20-byte sender information section (highlighted in Figure 2). Sender report is issued if the participant has sent at least one RTP packet during the last report period – otherwise a receiver report is issued.

Both the sender report and the receiver report include reception report blocks, one for each of the synchronization sources from which this participant has received RTP data packets since the last report. Reports are not issued for contributing sources listed in the CSRC list. Each reception report block provides statistics about the data received from the particular source indicated in that block.

<table>
<thead>
<tr>
<th>V</th>
<th>P</th>
<th>RC</th>
<th>PT=SR=200</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSRC of Sender</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NTP Timestamp, Most Significant Word</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NTP Timestamp, Least Significant Word</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RTP Timestamp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sender’s Packet Count</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sender’s Octet Count</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SSRC_n (SSRC of nᵗʰ Source)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fraction Lost</td>
<td>Cumulative Number of Packets Lost</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Extended Highest Sequence Number Received</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interarrival Jitter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Last SR (LSR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delay Since Last SR (DLSR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>…</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Profile-specific Extensions</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. RTCP Sender Report**

The first section of the sender report, called header, is 8 octets long.

- **Version (V, 2 bits)**
  
  Identifies the current version (which is the same for RTCP and RTP packets). The version defined in RFC 1889 is two.

- **Padding (P, 1 bit)**
  
  If the padding bit is set, this RTCP packet contains some additional padding octets at the end which are not part of the control information. The last octet of the packet contains the number of these padding octets.

- **Reception Report Count (RC, 5 bits)**
  
  The number of reception report blocks contained in this packet. A value of zero is valid.

- **Packet Type (PT, 8 bits)**
  
  Contains the constant 200 to identify this packet as an RTCP sender report.
• **Length (16 bits)**

Length of this RTCP packet in 32-bit words subtracted by one. (Includes the header and any padding.)

• **SSRC (32 bits)**

Synchronization source identifier for the originator of this sender report.

The second section, sender information, is 20 octets (five rows in Figure 2) long and it is present in all sender reports. It summarizes the data transmissions from this sender.

• **NTP Timestamp (64 bits)**

Indicates the wallclock time when this report was sent.

• **RTP Timestamp (32 bits)**

Corresponds to the same time as the NTP timestamp, but in the same units, and with the same random offset as the RTP timestamps in data packets. This correspondence may be used for intra- and inter-media synchronization for sources whose NTP timestamps are synchronized.

• **Sender’s Packet Count (32 bits)**

The total number of RTP data packets transmitted by the sender since starting transmission up until the time this sender report was generated. The count is reset if the sender changes its SSRC identifier.

• **Sender’s Octet Count (32 bits)**

The total number of payload octets (not including header or padding) transmitted in RTP data packets by the sender since the start of transmission up until the time this sender report was generated. The count is reset if the sender changes its SSRC identifier. This field can be used to estimate the average payload data rate.

The third section contains reception report blocks. The amount of these blocks depends on the number of other sources that this sender has been listening to since last report.

• **SSRC_n (Source Identifier, 32 bits)**

The SSRC identifier of the source that we are reporting about.

• **Fraction Lost (8 bits)**

The fraction of RTP data packets from source SSRC_n that were lost since the previous sender or receiver report was sent. If the loss is negative due to duplicates, the fraction lost is set to zero.

• **Cumulative Number of Packets Lost (24 bits)**

The total number of lost packets from source SSRC_n since the beginning of reception. This figure is defined to be the number of packets expected subtracted by the number of packets actually received. The number of packets received also includes late and duplicate packets. Thus packets that arrive late are not counted as lost, and the loss may be negative if there are duplicates. The number of packets expected is defined to be the last extended highest sequence number received subtracted by the initial sequence number received. This may be calculated as shown in RFC 1889 [Sch96a].

• **Extended Highest Sequence Number Received (32 bits)**

The low 16 bits contain the highest sequence number received in an RTP packet from source SSRC_n, and the most significant 16 bits extend that sequence number with the corresponding count of sequence number cycles.

• **Interarrival Jitter (32 bits)**

An estimation of the variance of the RTP packet interarrival time measured in timestamp units and expressed as an unsigned integer.

Interarrival jitter can be calculated as a difference in the relative transit time for two packets. The relative transit time is the difference between the packet’s RTP timestamp and the receiver’s clock at the time of arrival, measured in same units. If $S_i$ is the RTP timestamp of packet $i$ and $R_i$ is the time of arrival of packet $i$ (in RTP timestamp units), the difference in packet spacing for the two packets, $i$ and $j$, can be expressed as:

$$D_{(i,j)} = (R_j - R_i) - (S_j - S_i).$$

Interarrival jitter is updated each time when a packet is received from source SSRC_n (using the difference in packet spacing for that packet and the previous packet) according to the following formula:
When the reception report is issued, the current value of $J$ is sampled.

- **Last SR Timestamp (LSR, 32 bits)**
  The middle 32 bits of the NTP timestamp of the most recent RTCP sender report from source SSRC_n. If no sender report has been received yet, the field is set to zero.

- **Delay Since Last SR (DLSR, 32 bits)**
  The delay (expressed in NTP form: units of $1/65536$ seconds) since the last sender report arrived. Together with the last SR timestamp, the sender of this last SR can use it to compute the round trip time. If no SR packet has been received yet from SSRC_n, the DLSR field is set to zero.

**Receiver Report RTCP Packet**

Receiver report (RR) shares the same format with the sender report except that the packet type field contains the constant 201, and the five words of sender information are omitted (these are the NTP & RTP timestamps and sender’s packet & octet counts).

### 3 Header Compression

The high overhead of RTP/UDP/IP packets is a challenging issue, especially on slow links [Her00]. For example, a popular video-conferencing application, Microsoft NetMeeting [Net01], uses voice codec G.723.1, where a frame of 24 bytes is sent every 30 ms. This will produce a data rate of 6.3 kbit/s. Since RTP/UDP/IP headers add at least 40 bytes of overhead, and the link layer some bytes as well (PPP+HDLC add four bytes), the resulting bit rate will be over 18 kbit/s.

A common trade-off for reducing the bit rate is to put several frames in a single packet. However, this can set the conversational delay to a level that is far too high for most users. The overhead issue can be solved also by using header compression [Cas99].

Header compression is based on the simple idea that since most of the data packet overhead is constant for a given stream, it is possible to negotiate a shorter index for those constants (e.g. source and destination IP addresses and ports) when the stream is set up [Her00]. Other (variable) values can be reconstructed at the receiving end. To put it short: the sending host replaces the large RTP/UDP/IP header to a small index, and the receiving host reverses this operation. An RTP/UDP/IP header compression mechanism for low-speed links is described in RFC 2508. In many cases, all three headers can be compressed to 2-4 bytes. The compression is done on a link-by-link basis [Cas99].

#### 3.1 RFC 2508

The compression algorithm proposed in RFC 2508 draws heavily upon the design of TCP/IP header compression, which is described in RFC 1144 [Jac90].

**The Basic Idea**

In TCP/IP header compression, it has been observed that about half of the header bytes remain constant over the duration of the connection. After the header has been sent uncompressed once, the constant fields can be excluded from the following compressed headers. Headers can be further compressed with the help of differential coding on the changing fields.

In RTP header compression, some of the aforementioned techniques can be applied, but the major gain comes from the fact that although several fields change in every packet, the difference from packet to packet is often constant. If the compressor and decompressor maintain both the uncompressed header and the first-order differences, the only information that must be conveyed is an indication that the second-order difference was zero. If that is the case, the decompressor can reconstruct the original header without any loss of information by adding the first-order differences to the uncompressed header as each compressed packet is received.

**Header Compression for RTP/UDP/IP Packets**

In IPv4 header, only the total length, packet ID, and header check-sum fields typically change. The total length can be excluded, because it is provided by the link layer. Since the RFC 2508 compression scheme depends upon the link layer to provide good error detection, the header checksum may also be excluded.

In order to maintain lossless compression, changes in the packet ID are transmitted. The packet ID is usually incremented by one for each packet. In IPv6 base header, neither packet ID nor header checksum exist, and only the payload length field changes.

In UDP header, the length field is redundant with the IP total length field and the length indicated by the link layer. UDP checksum field will be zero in the case, where source does not generate any UDP checksums. Otherwise, the checksum must be sent intact in order to preserve lossless compression.
In most RTP headers, only the sequence number and timestamp change from packet to packet. If packets are not lost or misordered, the sequence number is incremented by one for each packet. For audio packets of constant duration, the timestamp is incremented by the number of sample periods conveyed in each packet.

If the second-order differences of the sequence number and timestamp fields are zero, the next packet header can be constructed from the previous header by adding the first-order differences (that are stored in the session context along with the uncompressed header) for these fields.

The marker bit is set on the first packet of an audio talkspurt. If it were treated as a constant field, such that each change would require sending the full RTP header, the compression would become quite inefficient. Because of this, one bit in the compressed header is reserved for the marker bit.

### 3.2 Other Proposals

There exist a number of other RTP/UDP/IP header compression mechanisms that have emerged after RFC 2508. They all should perform slightly better than the mechanism described in the RFC 2508. Some of the most recent proposals are Ericsson’s ROCCO [Lar00] and Nokia’s ACE [Khi00].

### 4 Playout Algorithms

#### 4.1 Playout Delay

In most packet audio applications, packets are buffered at the receiving host in order to compensate for variable network delay. The receiver buffer sizes can be constant or adaptively adjusted. Keeping the delay as small as possible, and avoiding excessive packet losses at the same time is not an easy task. The results of [Ram94] indicate that an adaptive algorithm, which explicitly adjusts to the sharp, spike-like increases in packet delay, can achieve a lower rate of lost packets.

Adaptive playout delay can be either per-talkspurt or per-packet based. In the former approach, playout delay remains constant throughout the talkspurt and the adjustments are done between talkspurts. The latter approach introduces gaps in speech, and thus it is not recommended for VoIP [Yle97].

Three different playout delay adjustment algorithms for packetized voice are presented in [Moo98]. The paper is focused on the tradeoff between packet playout delay and packet playout loss. The authors present an adaptive delay adjustment algorithm that tracks the network delay of recently received packets and maintains delay percentile information.

Some playout delay adjustment algorithms assume that the sender and receiver clocks are synchronized, but in [Moo98] this is not the case. The propagation delay is removed from end-to-end delay by subtracting out the minimum of measured end-to-end delays. Thus it is possible to concentrate on the variable delay component.

In the following section, we present a similar, although somewhat simpler, algorithm for adaptive playout delay adjustment. This algorithm does not assume synchronization of the sender and receiver clocks.

Waiting time in playout buffer is calculated with the following algorithm:

$$T_{\text{wait}, \, 0} = t_{\text{wait}}$$

For other packets, waiting time is calculated as follows:

$$T_{\text{wait}, \, i} = (T\text{Stamp}_i – T\text{Stamp}_{i-1}) – (\text{ReceivedAt}_i - \text{PlayAt}_{i-1})$$

If the result is negative, packet is discarded.

Whenever playout delay is adjusted, it will be the maximum of the initial playout delay and the current playout delay subtracted by the minimum $T_{\text{wait}}$ of the latest measurement period.

The following events will trigger the playout delay adjustment:

- If $N$ or more packets among the last $M$ packets (measurement period) arrive late, playout delay is adjusted upwards when the next talkspurt arrives.

- Similarly, if $M$ successive packets have been received all in time, playout delay is adjusted downwards before the next talkspurt.

Table 1 shows some simulation results for constant and adaptive playout delay. Network delay was simply modeled with exponential distribution with a mean of 30 milliseconds. Parameters used were: $N = 2$, $M = 100$, $t_{\text{wait}} = 100$ ms. Simulation duration was 200 seconds.

<table>
<thead>
<tr>
<th>Playout delay</th>
<th>Packet loss ratio</th>
<th>Mean</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>3.7%</td>
<td>100 ms</td>
<td>100 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>Adaptive</td>
<td>1.2%</td>
<td>150 ms</td>
<td>100 ms</td>
<td>240 ms</td>
</tr>
</tbody>
</table>
Simulation results show that there is a clear tradeoff between playout delay and playout loss. If we had selected a larger playout delay in the constant playout delay case, packet loss ratio would have been smaller. If we the variation of network delay is unknown, it can be very hard to set the constant playout delay. Simulation results also show that upper bound for adaptive playout delay is probably needed, because end-to-end delays longer than 400 milliseconds are not acceptable for voice [ITU00].

Figure 3 illustrates the simulated sequence of sent, received and synchronized VoIP packets, while Figure 4 illustrates the changes in playout delay.

Figure 3. Sent, Received and Synchronized Packets

4.2 Synchronization Delay and Clock Drift
Audio device driver is the link between the operating system and the hardware. Implementation of the device driver is a crucial factor in audio device performance. The driver can namely introduce unnecessary delay in both directions. If the audio device requests data at fixed intervals, that are not synchronized with the reception of incoming packets, an additional delay of half of the audio block duration is introduced [Sel01].

Clock drift means that the sampling and playout rates of the audio devices do not match. If the clocks at each end drift in different directions, buffer underruns at one end, and increased delay is experienced at the other end. Buffer underrun occurs when the receiver does not have anything to play. Modifying the playout rate by adding or dropping samples before the frame is transferred to the audio device can compensate for clock drift [Sel01].

5 VoIP Terminal Requirements and Commercial Implementations
According to [Kos98], PCs (or other terminals) that support real time interactive voice must have considerable processing power. The computational requirements of voice codecs increase with the voice compression ratio.

Microsoft NetMeeting [Net01] is a popular remote conferencing tool. The hardware requirements that have to be met in order to use the data, audio, and video features of NetMeeting are as follows:

- For Windows 95 or Windows 98: Pentium 90 processor with 16 MB of RAM
- For Windows NT: a Pentium 90 processor with 24 MB of RAM
- 56,000 bps or faster Internet connection
- Sound card with microphone and speakers
- Video capture card or camera that provides a Video for Windows capture driver

VocalTec Internet Phone Lite is mainly targeted for voice only connections. It has the following system requirements [Voc01]:

- Windows 95 or Windows NT 4.0 or higher
• Pentium 75 processor or higher
• 14,400 bps or faster Internet connection
• Sound card with microphone and speakers

6 Conclusions
RTP/RTCP protocol suite provides the means for real-time communication over IP by introducing time stamps and sequence numbers.

The amount of overhead in VoIP packets is a serious issue, especially on low-bandwidth links. This problem can be alleviated by header compression, which is done on link-by-link basis.

In order to make transmitted voice stream human understandable, playout buffering is needed to re-synchronize the stream. A new adaptive playout delay adjustment algorithm has been introduced. The operation of the algorithm is basically that if certain criteria are met, the delay between the sender and the receiver can be adaptively adjusted upwards or downwards.

Terminals that support real time interactive voice must have considerable processing power. The computational requirements of voice codecs increase with the voice compression ratio.

References


[ITU00] ITU-T Recommendation G.114 (05/00): One-way transmission time.


[Net01] Microsoft Netmeeting:


[Sch96b] H. Schulzrinne: RTP Profile for Audio and Video Conferences with Minimal Control (Request for Comments: 1890), January 1996.


[Voc01] VocalTec Internet Phone Lite: