Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-avt-rtp-new-01.ps

Audio/Video Transport Working Group
Schulzrinne/Casner/Frederick/Jacobson
Columbia U./Cisco/Xerox/LBNL

August 7, 1998
Expires: February 7, 1999

# RTP: A Transport Protocol for Real-Time Applications

## Status of this Memo



**Abstract**

This memorandum is a revision of RFC 1889 in preparation for advancement from Proposed Standard to Draft Standard status. Readers are encouraged to use the PostScript form of this draft to see where changes from RFC 1889 are marked by change bars. The revision process is not yet complete; some changes which have been discussed and tentatively accepted in meetings of the Audio/Video Transport working group have not yet been incorporated into this draft.

This memorandum describes RTP, the real-time transport protocol. RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of-service for real-time services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers. The protocol supports the use of RTP-level translators and mixers.

This specification is a product of the Audio/Video Transport working group within the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at `rem-conf@es.net` and/or the authors.

# Contents

## Resolution of Open Issues

[Note to the RFC Editor: This section is to be deleted when this draft is published as an RFC but is shown here for reference during the Last Call.]

Readers are directed to Appendix B, Changes from RFC 1889, for a listing of the changes that have been made in this draft. The changes are marked with change bars in the PostScript form of this draft.

The revisions in this draft are mostly complete for Working Group last call; the open issues have been addressed:

- A fudge factor has been added to the RTCP unconditional reconsideration algorithm to compensate for the fact that it settles to a steady state bandwidth that is below the desired level.
- A new "bin" mechanism has been added to the algorithm for sampled storaged of SSRC identifiers to avoid a temporary underestimate in group size when the group size is decreasing.
- The "reverse reconsideration" algorithm does not prevent the group size estimate from incorrectly dropping to zero for a short time when most participants of a large session leave at once but some remain. This has just been noted as only a secondary concern.
- Scaling of the minimum RTCP interval inversely proportional to the session bandwidth parameter has been added, but only in the direction of smaller intervals for higher bandwidth. Scaling to longer intervals for low bandwidths would cause a problem because this is an optional step. Some participants might be timed out prematurely if they scaled to a longer interval while others kept the nominal 5 seconds. The benefit of scaling longer was not considered great in any case.
- No change was specified for the jitter computation for media with several packets with the same timestamp. There is not a clear answer as to what should be done, or that any change would make a significant improvement.
- As proposed without objection at the Los Angeles IETF, definition of additional SDES items such as PHOTO URL and NICKNAME will be deferred to subsequent registration through IANA since that method has been established. This is in the spirit of minimizing changes to the protocol in the transition from Proposed to Draft.
- Nothing was added about allowing a translator to add its own random offsets to the sequence number and timestamp fields because it would likely cause more trouble than good.

# 1 Introduction

This memorandum specifies the real-time transport protocol (RTP), which provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. Those services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. However, RTP may be used with other suitable underlying network or transport protocols (see Section 10). RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.

Note that RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services to do so. It does *not* guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence.

While RTP is primarily designed to satisfy the needs of multi-participant multimedia conferences, it is not limited to that particular application. Storage of continuous data, interactive distributed simulation, active badge, and control and measurement applications may also find RTP applicable.

This document defines RTP, consisting of two closely-linked parts:

- the real-time transport protocol (RTP), to carry data that has real-time properties.
- the RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participants in an on-going session. The latter aspect of RTCP may be sufficient for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up, but it is not necessarily intended to support all of an application's control communication requirements. This functionality may be fully or partially subsumed by a separate session control protocol, which is beyond the scope of this document.

RTP represents a new style of protocol following the principles of application level framing and integrated layer processing proposed by Clark and Tennenhouse [1]. That is, RTP is intended to be malleable to provide the information required by a particular application and will often be integrated into the application processing rather than being implemented as a separate layer. RTP is a protocol framework that is deliberately not complete. This document specifies those functions expected to be common across all the applications for which RTP would be appropriate. Unlike conventional protocols in which additional functions might be accommodated by making the protocol more general or by adding an option mechanism that would require parsing, RTP is intended to be tailored through modifications and/or additions to the headers as needed. Examples are given in Sections 5.3 and 6.4.3.

Therefore, in addition to this document, a complete specification of RTP for a particular application will require one or more companion documents (see Section 12):

- a *profile* specification document, which defines a set of payload type codes and their mapping to payload formats (e.g., media encodings). A profile may also define extensions or modifications to RTP that are specific to a particular class of applications. Typically an application will operate under only one profile. A profile for audio and video data may be found in the companion RFC 1890.
- *payload format* specification documents, which define how a particular payload, such as an audio or video encoding, is to be carried in RTP.

A discussion of real-time services and algorithms for their implementation as well as background discussion on some of the RTP design decisions can be found in [2].

## 1.1 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 [3] and indicate requirement levels for compliant RTP implementations.

# 2 RTP Use Scenarios

The following sections describe some aspects of the use of RTP. The examples were chosen to illustrate the basic operation of applications using RTP, not to limit what RTP may be used for. In these examples, RTP is carried on top of IP and UDP, and follows the conventions established by the profile for audio and video specified in the companion RFC 1890 (updated by Internet-Draft `draft-ietf-avt-profile-new`).

## 2.1 Simple Multicast Audio Conference

A working group of the IETF meets to discuss the latest protocol draft, using the IP multicast services of the Internet for voice communications. Through some allocation mechanism the working group chair obtains a multicast group address and pair of ports. One port is used for audio data, and the other is used for control (RTCP) packets. This address and port information is distributed to the intended participants. If privacy is desired, the data and control packets may be encrypted as specified in Section 9.1, in which case an encryption key must also be generated and distributed. The exact details of these allocation and distribution mechanisms are beyond the scope of RTP.

The audio conferencing application used by each conference participant sends audio data in small chunks of, say, 20 ms duration. Each chunk of audio data is preceded by an RTP header; RTP header and data are in turn contained in a UDP packet. The RTP header indicates what type of audio encoding (such as PCM, ADPCM or LPC) is contained in each packet so that senders can change the encoding during a conference, for example, to accommodate a new participant that is connected through a low-bandwidth link or react to indications of network congestion.

The Internet, like other packet networks, occasionally loses and reorders packets and delays them by variable amounts of time. To cope with these impairments, the RTP header contains timing information and a sequence number that allow the receivers to reconstruct the timing produced by the source, so that in this example, chunks of audio are contiguously played out the speaker every 20 ms. This timing reconstruction is performed separately for each source of RTP packets in the conference. The sequence number can also be used by the receiver to estimate how many packets are being lost.

Since members of the working group join and leave during the conference, it is useful to know who is participating at any moment and how well they are receiving the audio data. For that purpose, each instance of the audio application in the conference periodically multicasts a reception report plus the name of its user on the RTCP (control) port. The reception report indicates how well the current speaker is being received and may be used to control adaptive encodings. In addition to the user name, other identifying information may also be included subject to control bandwidth limits. A site sends the RTCP BYE packet (Section 6.6) when it leaves the conference.

## 2.2 Audio and Video Conference

If both audio and video media are used in a conference, they are transmitted as separate *RTP sessions*. That is, separate RTP and RTCP packets are transmitted for each medium using two different UDP port pairs and/or multicast addresses. There is no direct coupling at the RTP level between the audio and video sessions, except that a user participating in both sessions should use the same distinguished (canonical) name in the RTCP packets for both so that the sessions can be associated.

One motivation for this separation is to allow some participants in the conference to receive only one medium if they choose. Further explanation is given in Section 5.2. Despite the separation, synchronized playback of a source's audio and video can be achieved using timing information carried in the RTCP packets for both sessions.

### 2.3 Mixers and Translators

So far, we have assumed that all sites want to receive media data in the same format. However, this may not always be appropriate. Consider the case where participants in one area are connected through a low-speed link to the majority of the conference participants who enjoy high-speed network access. Instead of forcing everyone to use a lower-bandwidth, reduced-quality audio encoding, an RTP-level relay called a *mixer* may be placed near the low-bandwidth area. This mixer resynchronizes incoming audio packets to reconstruct the constant 20 ms spacing generated by the sender, mixes these reconstructed audio streams into a single stream, translates the audio encoding to a lower-bandwidth one and forwards the lower-bandwidth packet stream across the low-speed link. These packets might be unicast to a single recipient or multicast on a different address to multiple recipients. The RTP header includes a means for mixers to identify the sources that contributed to a mixed packet so that correct talker indication can be provided at the receivers.

Some of the intended participants in the audio conference may be connected with high bandwidth links but might not be directly reachable via IP multicast. For example, they might be behind an application-level firewall that will not let any IP packets pass. For these sites, mixing may not be necessary, in which case another type of RTP-level relay called a *translator* may be used. Two translators are installed, one on either side of the firewall, with the outside one funneling all multicast packets received through a secure connection to the translator inside the firewall. The translator inside the firewall sends them again as multicast packets to a multicast group restricted to the site's internal network.

Mixers and translators may be designed for a variety of purposes. An example is a video mixer that scales the images of individual people in separate video streams and composites them into one video stream to simulate a group scene. Other examples of translation include the connection of a group of hosts speaking only IP/UDP to a group of hosts that understand only ST-II, or the packet-by-packet encoding translation of video streams from individual sources without resynchronization or mixing. Details of the operation of mixers and translators are given in Section 7.

### 2.4 Layered Encodings

Multimedia applications should be able to adjust the transmission rate to match the capacity of the receiver or to adapt to network congestion. Many implementations place the responsibility of rate-adaptivity at the source. This does not work well with multicast transmission because of the conflicting bandwidth requirements of heterogeneous receivers. The result is often a least-common denominator scenario, where the smallest pipe in the network mesh dictates the quality and fidelity of the overall live multimedia "broadcast".

Instead, responsibility for rate-adaptation can be placed at the receivers by combining a layered encoding with a layered transmission system. In the context of RTP over IP multicast, the source can stripe the progressive layers of a hierarchically represented signal across multiple RTP sessions each carried on its own multicast group. Receivers can then adapt to network heterogeneity and control their reception bandwidth by joining only the appropriate subset of the multicast groups.

Details of the use of RTP with layered encodings are given in Sections 6.3.9, 8.3 and 10.

# 3 Definitions

**RTP payload:** The data transported by RTP in a packet, for example audio samples or compressed video data. The payload format and interpretation are beyond the scope of this document.

**RTP packet:** A data packet consisting of the fixed RTP header, a possibly empty list of contributing sources (see below), and the payload data. Some underlying protocols may require an encapsulation of the RTP packet to be defined. Typically one packet of the underlying protocol contains a single RTP packet, but several RTP packets may be contained if permitted by the encapsulation method (see Section 10).

**RTCP packet:** A control packet consisting of a fixed header part similar to that of RTP data packets, followed by structured elements that vary depending upon the RTCP packet type. The formats are defined in Section 6. Typically, multiple RTCP packets are sent together as a compound RTCP packet in a single packet of the underlying protocol; this is enabled by the length field in the fixed header of each RTCP packet.

**Port:** The "abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. TCP/IP protocols identify ports using small positive integers." [4] The transport selectors (TSEL) used by the OSI transport layer are equivalent to ports. RTP depends upon the lower-layer protocol to provide some mechanism such as ports to multiplex the RTP and RTCP packets of a session.

**Transport address:** The combination of a network address and port that identifies a transport-level endpoint, for example an IP address and a UDP port. Packets are transmitted from a source transport address to a destination transport address.

**RTP media type:** An RTP media type is the collection of payload types which can be carried within a single RTP session. The RTP Profile assigns RTP media types to RTP payload types.

**RTP session:** The association among a set of participants communicating with RTP. For each participant, the session is defined by a particular pair of destination transport addresses (one network address plus a port pair for RTP and RTCP). The destination transport address pair may be common for all participants, as in the case of IP multicast, or may be different for each, as in the case of individual unicast network addresses and port pairs. In a multimedia session, each medium is carried in a separate RTP session with its own RTCP packets. The multiple RTP sessions are distinguished by different port number pairs and/or different multicast addresses.

**Synchronization source (SSRC):** The source of a stream of RTP packets, identified by a 32-bit numeric SSRC identifier carried in the RTP header so as not to be dependent upon the network address. All packets from a synchronization source form part of the same timing and sequence number space, so a receiver groups packets by synchronization source for playback. Examples of synchronization sources include the sender of a stream of packets derived from a signal source such as a microphone or a camera, or an RTP mixer (see below). A synchronization source may change its data format, e.g., audio encoding, over time. The SSRC identifier is a randomly chosen value meant to be globally unique within a particular RTP session (see Section 8). A participant need not use the same SSRC identifier for all the RTP sessions in a multimedia session; the binding of the SSRC identifiers is provided through RTCP (see Section 6.5.1). If a participant generates multiple streams in one RTP session, for example from separate video cameras, each must be identified as a different SSRC.

**Contributing source (CSRC):** A source of a stream of RTP packets that has contributed to the combined stream produced by an RTP mixer (see below). The mixer inserts a list of the SSRC identifiers of the sources that contributed to the generation of a particular packet into the RTP header of that packet. This list is called the CSRC list. An example application is audio conferencing where a mixer indicates all the talkers whose speech was combined to produce the outgoing packet, allowing the receiver to indicate the current talker, even though all the audio packets contain the same SSRC identifier (that of the mixer).

**End system:** An application that generates the content to be sent in RTP packets and/or consumes the content of received RTP packets. An end system can act as one or more synchronization sources in a particular RTP session, but typically only one.

**Mixer:** An intermediate system that receives RTP packets from one or more sources, possibly changes the data format, combines the packets in some manner and then forwards a new RTP packet. Since the timing among multiple input sources will not generally be synchronized, the mixer will make timing adjustments among the streams and generate its own timing for the combined stream. Thus, all data packets originating from a mixer will be identified as having the mixer as their synchronization source.

**Translator:** An intermediate system that forwards RTP packets with their synchronization source identifier intact. Examples of translators include devices that convert encodings without mixing, replicators from multicast to unicast, and application-level filters in firewalls.

**Monitor:** An application that receives RTCP packets sent by participants in an RTP session, in particular the reception reports, and estimates the current quality of service for distribution monitoring, fault diagnosis and long-term statistics. The monitor function is likely to be built into the application(s) participating in the session, but may also be a separate application that does not otherwise participate and does not send or receive the RTP data packets. These are called *third party* monitors.

**Non-RTP means:** Protocols and mechanisms that may be needed in addition to RTP to provide a usable service. In particular, for multimedia conferences, a conference control application may distribute multicast addresses and keys for encryption, negotiate the encryption algorithm to be used, and define dynamic mappings between RTP payload type values and the payload formats they represent for formats that do not have a predefined payload type value. For simple applications, electronic mail or a conference database may also be used. The specification of such protocols and mechanisms is outside the scope of this document.

# 4 Byte Order, Alignment, and Time Format

All integer fields are carried in network byte order, that is, most significant byte (octet) first. This byte order is commonly known as big-endian. The transmission order is described in detail in [5, Appendix A]. Unless otherwise noted, numeric constants are in decimal (base 10).

All header data is aligned to its natural length, i.e., 16-bit fields are aligned on even offsets, 32-bit fields are aligned at offsets divisible by four, etc. Octets designated as padding have the value zero.

Wallclock time (absolute date and time) is represented using the timestamp format of the Network Time Protocol (NTP), which is in seconds relative to 0h UTC on 1 January 1900 [6]. The full resolution NTP timestamp is a 64-bit unsigned fixed-point number with the integer part in the first 32 bits and the fractional part in the last 32 bits. In some fields where a more compact representation is appropriate, only the middle 32 bits are used; that is, the low 16 bits of the integer part and the high 16 bits of the fractional part. The high 16 bits of the integer part must be determined independently.

# 5 RTP Data Transfer Protocol

## 5.1 RTP Fixed Header Fields

The RTP header has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The first twelve octets are present in every RTP packet, while the list of CSRC identifiers is present only when inserted by a mixer. The fields have the following meaning:

**version (V):** 2 bits
This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)

**padding (P):** 1 bit
If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself. Padding may be needed by some encryption algorithms with fixed block sizes or for carrying several RTP packets in a lower-layer protocol data unit.

**extension (X):** 1 bit
If the extension bit is set, the fixed header is followed by exactly one header extension, with a format defined in Section 5.3.1.

**CSRC count (CC):** 4 bits
The CSRC count contains the number of CSRC identifiers that follow the fixed header.

**marker (M):** 1 bit
The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream. A profile may define additional marker bits or specify that there is no marker bit by changing the number of bits in the payload type field (see Section 5.3).

**payload type (PT):** 7 bits
This field identifies the format of the RTP payload and determines its interpretation by the application. A profile specifies a default static mapping of payload type codes to payload formats. Additional payload type codes may be defined dynamically through non-RTP means (see Section 3). An initial set of

default mappings for audio and video is specified in the companion RFC 1890 (updated by Internet-Draft `draft-ietf-avt-profile-new`), and may be extended in future editions of the Assigned Numbers RFC [7]. An RTP sender emits a single RTP payload type at any given time; this field is not intended for multiplexing separate media streams (see Section 5.2).

A receiver MUST ignore packets with payload types that it does not understand.

**sequence number:** 16 bits
The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number SHOULD be random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt according to the method in Section 9.1, because the packets may flow through a translator that does. Techniques for choosing unpredictable numbers are discussed in [8].

**timestamp:** 32 bits
The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations (see Section 6.4.1). The resolution of the clock must be sufficient for the desired synchronization accuracy and for measuring packet arrival jitter (one tick per video frame is typically not sufficient). The clock frequency is dependent on the format of data carried as payload and is specified statically in the profile or payload format specification that defines the format, or may be specified dynamically for payload formats defined through non-RTP means. If RTP packets are generated periodically, the nominal sampling instant as determined from the sampling clock is to be used, not a reading of the system clock. As an example, for fixed-rate audio the timestamp clock would likely increment by one for each sampling period. If an audio application reads blocks covering 160 sampling periods from the input device, the timestamp would be increased by 160 for each such block, regardless of whether the block is transmitted in a packet or dropped as silent.

The initial value of the timestamp is random, as for the sequence number. Several consecutive RTP packets may have equal timestamps if they are (logically) generated at once, e.g., belong to the same video frame. Consecutive RTP packets may contain timestamps that are not monotonic if the data is not transmitted in the order it was sampled, as in the case of MPEG interpolated video frames. (The sequence numbers of the packets as transmitted will still be monotonic.)

**SSRC:** 32 bits
The SSRC field identifies the synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier. An example algorithm for generating a random identifier is presented in Appendix A.6. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve collisions. Section 8 describes the probability of collision along with a mechanism for resolving collisions and detecting RTP-level forwarding loops based on the uniqueness of the SSRC identifier. If a source changes its source transport address, it must also choose a new SSRC identifier to avoid being interpreted as a looped source (see Section 8.2).

**CSRC list:** 0 to 15 items, 32 bits each
The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 may be

identified. CSRC identifiers are inserted by mixers, using the SSRC identifiers of contributing sources. For example, for audio packets the SSRC identifiers of all sources that were mixed together to create a packet are listed, allowing correct talker indication at the receiver.

### 5.2 Multiplexing RTP Sessions

For efficient protocol processing, the number of multiplexing points should be minimized, as described in the integrated layer processing design principle [1]. In RTP, multiplexing is provided by the destination transport address (network address and port number) which define an RTP session. For example, in a teleconference composed of audio and video media encoded separately, each medium should be carried in a separate RTP session with its own destination transport address. It is *not* intended that the audio and video streams be carried in a single RTP session and demultiplexed based on the payload type or SSRC fields. Interleaving packets with different RTP media types but using the same SSRC would introduce several problems:

1. If, say, two audio streams shared the same RTP session and the same SSRC value, and one were to change encodings and thus acquire a different RTP payload type, there would be no general way of identifying which stream had changed encodings.

2. An SSRC is defined to identify a single timing and sequence number space. Interleaving multiple payload types would require different timing spaces if the media clock rates differ and would require different sequence number spaces to tell which payload type suffered packet loss.

3. The RTCP sender and receiver reports (see Section 6.4) can only describe one timing and sequence number space per SSRC and do not carry a payload type field.

4. An RTP mixer would not be able to combine interleaved streams of incompatible media into one stream.

5. Carrying multiple media in one RTP session precludes: the use of different network paths or network resource allocations if appropriate; reception of a subset of the media if desired, for example just audio if video would exceed the available bandwidth; and receiver implementations that use separate processes for the different media, whereas using separate RTP sessions permits either single- or multiple-process implementations.

Using a different SSRC for each medium but sending them in the same RTP session would avoid the first three problems but not the last two.

### 5.3 Profile-Specific Modifications to the RTP Header

The existing RTP data packet header is believed to be complete for the set of functions required in common across all the application classes that RTP might support. However, in keeping with the ALF design principle, the header may be tailored through modifications or additions defined in a profile specification while still allowing profile-independent monitoring and recording tools to function.

- The marker bit and payload type field carry profile-specific information, but they are allocated in the fixed header since many applications are expected to need them and might otherwise have to add another 32-bit word just to hold them. The octet containing these fields may be redefined by a profile to suit different requirements, for example with a more or fewer marker bits. If there are any marker bits, one should be located in the most significant bit of the octet since profile-independent monitors may be able to observe a correlation between packet loss patterns and the marker bit.

- Additional information that is required for a particular payload format, such as a video encoding, should be carried in the payload section of the packet. This might be in a header that is always present at the start of the payload section, or might be indicated by a reserved value in the data pattern.
- If a particular class of applications needs additional functionality independent of payload format, the profile under which those applications operate should define additional fixed fields to follow immediately after the SSRC field of the existing fixed header. Those applications will be able to quickly and directly access the additional fields while profile-independent monitors or recorders can still process the RTP packets by interpreting only the first twelve octets.

If it turns out that additional functionality is needed in common across all profiles, then a new version of RTP should be defined to make a permanent change to the fixed header.

### 5.3.1 RTP Header Extension

An extension mechanism is provided to allow individual implementations to experiment with new payload-format-independent functions that require additional information to be carried in the RTP data packet header. This mechanism is designed so that the header extension may be ignored by other interoperating implementations that have not been extended.

Note that this header extension is intended only for limited use. Most potential uses of this mechanism would be better done another way, using the methods described in the previous section. For example, a profile-specific extension to the fixed header is less expensive to process because it is not conditional nor in a variable location. Additional information required for a particular payload format should *not* use this header extension, but should be carried in the payload section of the packet.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      defined by profile       |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        header extension                       |
|                             ....                              |
```

If the X bit in the RTP header is one, a variable-length header extension is appended to the RTP header, following the CSRC list if present. The header extension contains a 16-bit length field that counts the number of 32-bit words in the extension, excluding the four-octet extension header (therefore zero is a valid length). Only a single extension may be appended to the RTP data header. To allow multiple interoperating implementations to each experiment independently with different header extensions, or to allow a particular implementation to experiment with more than one type of header extension, the first 16 bits of the header extension are left open for distinguishing identifiers or parameters. The format of these 16 bits is to be defined by the profile specification under which the implementations are operating. This RTP specification does not define any header extensions itself.

## 6 RTP Control Protocol — RTCP

The RTP control protocol (RTCP) is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. The underlying protocol must provide multiplexing of the data and control packets, for example using separate port numbers with UDP. RTCP performs four functions:

1. The primary function is to provide feedback on the quality of the data distribution. This is an integral part of the RTP's role as a transport protocol and is related to the flow and congestion control functions of other transport protocols. The feedback may be directly useful for control of adaptive encodings [9, 10], but experiments with IP multicasting have shown that it is also critical to get feedback from the receivers to diagnose faults in the distribution. Sending reception feedback reports to all participants allows one who is observing problems to evaluate whether those problems are local or global. With a distribution mechanism like IP multicast, it is also possible for an entity such as a network service provider who is not otherwise involved in the session to receive the feedback information and act as a third-party monitor to diagnose network problems. This feedback function is performed by the RTCP sender and receiver reports, described below in Section 6.4.

2. RTCP carries a persistent transport-level identifier for an RTP source called the *canonical name* or CNAME, Section 6.5.1. Since the SSRC identifier may change if a conflict is discovered or a program is restarted, receivers require the CNAME to keep track of each participant. Receivers may also require the CNAME to associate multiple data streams from a given participant in a set of related RTP sessions, for example to synchronize audio and video. Inter-media synchronization also requires the NTP and RTP timestamps included in RTCP packets by data senders.

3. The first two functions require that all participants send RTCP packets, therefore the rate must be controlled in order for RTP to scale up to a large number of participants. By having each participant send its control packets to all the others, each can independently observe the number of participants. This number is used to calculate the rate at which the packets are sent, as explained in Section 6.2.

4. A fourth, optional function is to convey minimal session control information, for example participant identification to be displayed in the user interface. This is most likely to be useful in "loosely controlled" sessions where participants enter and leave without membership control or parameter negotiation. RTCP serves as a convenient channel to reach all the participants, but it is not necessarily expected to support all the control communication requirements of an application. A higher-level session control protocol, which is beyond the scope of this document, may be needed.

Functions 1-3 SHOULD be used in all environments, but particularly in the IP multicast environment. RTP application designers SHOULD avoid mechanisms that can only work in unicast mode and will not scale to larger numbers. Transmission of RTCP MAY be controlled separately for senders and receivers for cases such as unidirectional links where feedback from receivers is not possible.

### 6.1 RTCP Packet Format

This specification defines several RTCP packet types to carry a variety of control information:

**SR:** Sender report, for transmission and reception statistics from participants that are active senders

**RR:** Receiver report, for reception statistics from participants that are not active senders

**SDES:** Source description items, including CNAME

**BYE:** Indicates end of participation

**APP:** Application specific functions

Each RTCP packet begins with a fixed part similar to that of RTP data packets, followed by structured elements that may be of variable length according to the packet type but always end on a 32-bit boundary. The alignment requirement and a length field in the fixed part of each packet are included to make RTCP packets "stackable". Multiple RTCP packets may be concatenated without any intervening separators to form a *compound RTCP packet* that is sent in a single packet of the lower layer protocol, for example UDP. There is no explicit count of individual RTCP packets in the compound packet since the lower layer protocols are expected to provide an overall length to determine the end of the compound packet.

Each individual RTCP packet in the compound packet may be processed independently with no requirements upon the order or combination of packets. However, in order to perform the functions of the protocol, the following constraints are imposed:

- Reception statistics (in SR or RR) should be sent as often as bandwidth constraints will allow to maximize the resolution of the statistics, therefore each periodically transmitted compound RTCP packet should include a report packet.

- New receivers need to receive the CNAME for a source as soon as possible to identify the source and to begin associating media for purposes such as lip-sync, so each compound RTCP packet should also include the SDES CNAME.

- The number of packet types that may appear first in the compound packet should be limited to increase the number of constant bits in the first word and the probability of successfully validating RTCP packets against misaddressed RTP data packets or other unrelated packets.

Thus, all RTCP packets MUST be sent in a compound packet of at least two individual packets, with the following format recommended:

**Encryption prefix:** If and only if the compound packet is to be encrypted according to the method in Section 9.1, it MUST be prefixed by a random 32-bit quantity redrawn for every compound packet transmitted. If padding is required for the encryption, it MUST be added to the last packet of the compound packet.

**SR or RR:** The first RTCP packet in the compound packet MUST always be a report packet to facilitate header validation as described in Appendix A.2. This is true even if no data has been sent nor received, in which case an empty RR is sent, and even if the only other RTCP packet in the compound packet is a BYE.

**Additional RRs:** If the number of sources for which reception statistics are being reported exceeds 31, the number that will fit into one SR or RR packet, then additional RR packets should follow the initial report packet.

**SDES:** An SDES packet containing a CNAME item must be included in each compound RTCP packet. Other source description items may optionally be included if required by a particular application, subject to bandwidth constraints (see Section 6.3.9).

**BYE or APP:** Other RTCP packet types, including those yet to be defined, may follow in any order, except that BYE should be the last packet sent with a given SSRC/CSRC. Packet types may appear more than once.

It is advisable for translators and mixers to combine individual RTCP packets from the multiple sources they are forwarding into one compound packet whenever feasible in order to amortize the packet overhead (see Section 7). An example RTCP compound packet as might be produced by a mixer is shown in Fig. 1. If the overall length of a compound packet would exceed the maximum transmission unit (MTU) of the network path, it may be segmented into multiple shorter compound packets to be transmitted in separate packets of the underlying protocol. Note that each of the compound packets must begin with an SR or RR packet.

An implementation may ignore incoming RTCP packets with types unknown to it. Additional RTCP packet types may be registered with the Internet Assigned Numbers Authority (IANA).

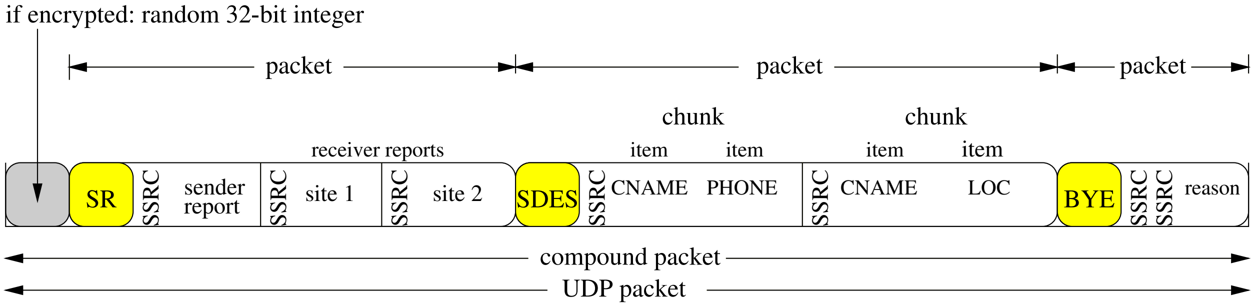


Figure 1: Example of an RTCP compound packet

## 6.2 RTCP Transmission Interval

RTP is designed to allow an application to scale automatically over session sizes ranging from a few participants to thousands. For example, in an audio conference the data traffic is inherently self-limiting because only one or two people will speak at a time, so with multicast distribution the data rate on any given link remains relatively constant independent of the number of participants. However, the control traffic is not self-limiting. If the reception reports from each participant were sent at a constant rate, the control traffic would grow linearly with the number of participants. Therefore, the rate must be scaled down by dynamically calculating the interval between RTCP packet transmissions.

For each session, it is assumed that the data traffic is subject to an aggregate limit called the "session bandwidth" to be divided among the participants. This bandwidth might be reserved and the limit enforced by the network. If there is no reservation, there may be other constraints, depending on the environment, that establish the "reasonable" maximum for the session to use, and that would be the session bandwidth. The session bandwidth may be chosen based or some cost or *a priori* knowledge of the available network bandwidth for the session. It is somewhat independent of the media encoding, but the encoding choice may be limited by the session bandwidth. Often, the session bandwidth is the sum of the nominal bandwidths of the senders expected to be concurrently active. For teleconference audio, this number would typically be one sender's bandwidth. For layered encodings, each layer is a separate RTP session with its own session bandwidth parameter.

The session bandwidth parameter is expected to be supplied by a session management application when it invokes a media application, but media applications may also set a default based on the single-sender data bandwidth for the encoding selected for the session. The application may also enforce bandwidth limits based on multicast scope rules or other criteria.

Bandwidth calculations for control and data traffic include lower-layer transport and network protocols (e.g., UDP and IP) since that is what the resource reservation system would need to know. The application can also be expected to know which of these protocols are in use. Link level headers are not included in the calculation since the packet will be encapsulated with different link level headers as it travels.

The control traffic should be limited to a small and known fraction of the session bandwidth: small so that the primary function of the transport protocol to carry data is not impaired; known so that the control traffic can be included in the bandwidth specification given to a resource reservation protocol, and so that each participant can independently calculate its share. It is RECOMMENDED that the fraction of the session bandwidth allocated to RTCP be fixed at 5%. It is also RECOMMENDED that 1/4 of the RTCP bandwidth be dedicated to participants that are sending data so that in sessions with a large number of receivers but a small number of senders, newly joining participants will more quickly receive the CNAME for the sending sites. When the proportion of senders is greater than 1/4 of the participants, the senders get their proportion of the full RTCP bandwidth. While the values of these and other constants in the interval calculation are not critical, all participants in the session MUST use the same values so the same interval will be calculated. Therefore, these constants should be fixed for a particular profile.

A profile MAY specify that the control traffic bandwidth may be a separate parameter of the session rather than a strict percentage of the session bandwidth. Using a separate parameter allows rate-adaptive applications to set an RTCP bandwidth consistent with a "typical" data bandwidth that is lower than the maximum bandwidth specified by the session bandwidth parameter.

The profile MAY further specify that the control traffic bandwidth may be divided into two separate session parameters for those participants which are active data senders and those which are not. Following the recommendation that 1/4 of the RTCP bandwidth be dedicated to data senders, the RECOMMENDED default values for these two parameters would be 1.25% and 3.75%, respectively. When the proportion of senders is greater than 1/4 of the participants, the senders get their proportion of the sum of these parameters. Using two parameters allows RTCP reception reports to be turned off entirely for a particular session by setting the RTCP bandwidth for non-data-senders to zero while keeping the RTCP bandwidth for data senders non-zero so that sender reports can still be sent for inter-media synchronization. This may be appropriate for systems operating on unidirectional links or for sessions that don't require feedback on the quality of reception.

The calculated interval between transmissions of compound RTCP packets SHOULD also have a lower bound to avoid having bursts of packets exceed the allowed bandwidth when the number of participants is small and the traffic isn't smoothed according to the law of large numbers. It also keeps the report interval from becoming too small during transient outages like a network partition such that adaptation is delayed when the partition heals. At application startup, a delay SHOULD be imposed before the first compound RTCP packet is sent to allow time for RTCP packets to be received from other participants so the report interval will converge to the correct value more quickly. This delay MAY be set to half the minimum interval to allow quicker notification that the new participant is present. The RECOMMENDED value for a fixed minimum interval is 5 seconds.

An implementation MAY scale the minimum RTCP interval to a smaller value inversely proportional to the session bandwidth parameter with the following limitations:

- For multicast sessions, only active data senders MAY use the reduced minimum value to calculate the interval for transmission of compound RTCP packets.

- For unicast sessions, the reduced value MAY be used by participants that are not active data senders as well, and the delay before sending the initial compound RTCP packet may be zero.

- For all sessions, the fixed minimum SHOULD be used when calculating the participant timeout interval (see Section 6.3.5 so that implementations which do not to use the reduced value for transmitting RTCP packets are not timed out by other participants prematurely.
- The RECOMMENDED value for the reduced minimum in seconds is 360 divided by the session bandwidth in kilobits/second. This minimum is smaller than 5 seconds for bandwidths greater than 72 kb/s.

The algorithm described in Section 6.3 and Appendix A.7 was designed to meet the goals outlined above. It calculates the interval between sending compound RTCP packets to divide the allowed control traffic bandwidth among the participants. This allows an application to provide fast response for small sessions where, for example, identification of all participants is important, yet automatically adapt to large sessions. The algorithm incorporates the following characteristics:

- The calculated interval between RTCP packets scales linearly with the number of members in the group. It is this linear factor which allows for a constant amount of control traffic when summed across all members.
- The interval between RTCP packets is varied randomly over the range [0.5,1.5] times the calculated interval to avoid unintended synchronization of all participants [11]. The first RTCP packet sent after joining a session is also delayed by a random variation of half the minimum RTCP interval.
- A dynamic estimate of the average compound RTCP packet size is calculated, including all those received and sent, to automatically adapt to changes in the amount of control information carried.
- Since the calculated interval is dependent on the number of observed group members, there may be undesirable startup effects when a new user joins an existing session, or many users simultaneously join a new session. These new users will initially have incorrect estimates of the group membership, and thus their RTCP transmission interval will be too short. This problem can be significant if many users join the session simultaneously. To deal with this, an algorithm called "timer reconsideration" is employed. This algorithm implements a simple back-off mechanism which causes users to hold back RTCP packet transmission if the group sizes are increasing.
- When users leave a session, either with a BYE or by timeout, the group membership decreases, and thus the calculated interval should decrease. A "reverse reconsideration" algorithm is used to allow members to more quickly reduce their intervals in response to group membership decreases.
- BYE packets are given different treatment than other RTCP packets. When a user leaves a group, and wishes to send a BYE packet, it may do so before its next scheduled RTCP packet. However, transmission of BYE's follows a back-off algorithm which avoids floods of BYE packets should a large number of members simultaneously leave the session.

This algorithm may be used for sessions in which all participants are allowed to send. In that case, the session bandwidth parameter is the product of the individual sender's bandwidth times the number of participants, and the RTCP bandwidth is 5% of that.

Details of the algorithm's operation are given in the sections that follow. Appendix A.7 gives an example implementation.

### 6.2.1 Maintaining the number of session members

Calculation of the RTCP packet interval depends upon an estimate of the number of sites participating in the session. New sites are added to the count when they are heard, and an entry for each SHOULD be created in a table indexed by the SSRC or CSRC identifier (see Section 8.2) to keep track of them. New entries MAY be considered not valid until multiple packets carrying the new SSRC have been received (see Appendix A.1). Entries MAY be deleted from the table when an RTCP BYE packet with the corresponding SSRC identifier is received, except that some straggler data packets might arrive after the BYE and cause the entry to be recreated. Instead, the entry should be marked as having received a BYE and then deleted after an appropriate delay.

A participant may mark another site inactive, or delete it if not yet valid, if no RTP or RTCP packet has been received for a small number of RTCP report intervals (5 is suggested). This provides some robustness against packet loss. All sites must calculate roughly the same value for the RTCP report interval in order for this timeout to work properly.

Once a site has been validated, then if it is later marked inactive the state for that site should still be retained and the site should continue to be counted in the total number of sites sharing RTCP bandwidth for a period long enough to span typical network partitions. This is to avoid excessive traffic, when the partition heals, due to an RTCP report interval that is too small. A timeout of 30 minutes is suggested. Note that this is still larger than 5 times the largest value to which the RTCP report interval is expected to usefully scale, about 2-5 minutes.

For sessions with a very large number of participants, it may be impractical to maintain a table to store the SSRC identifier and state information for all of them. An implementation MAY use SSRC sampling, as described here, to reduce the storage requirements. An implementation MAY use any other algorithm with similar performance. A key requirement is that any algorithm considered SHOULD NOT substantially underestimate the group size, although it MAY overestimate.

The sampling algorithm employs a mask with the $m$ least significant bits set to one and uses the participant's own SSRC identifier as a (random) key. If a newly received SSRC matches the key when both are ANDed with the mask, the new SSRC identifier is added to the table, otherwise it is ignored. An exception is that the SSRC identifiers of data senders must be maintained in the table even when their SSRC does not match under the masking operation because the potentially small number of senders must be accurate for the RTCP interval calculation. Initially, the mask starts with $m = 0$, so that every SSRC identifier is accepted and placed into the table. When the number of table entries reaches some threshold, $B$, $m$ is increased by 1 bit, and all the SSRC identifiers in the table which no longer match under the mask are discarded. This will reduce the table size by roughly half. As the group size continues to increase, the user MAY further increase the mask size by an additional bit when the table size once again approaches the threshold. An implementation MUST maintain a table that can accomodate at least $B = 100$ users, for reasonable statistical accuracy.

The algorithm also maintains a set of 32 bins, numbered 0 through 31. When a new participant shows up whose SSRC matches the key under the current mask (with $m$ bits), the SSRC identifier is placed in bin in bin $m$ that still match under the $m+1$ bit mask are moved from bin $m$ to bin $m+1$, otherwise they are discarded as mentioned previously. The SSRC identifiers of data senders are always kept and are always placed in the 0th bin. When a sender stops sending, its SSRC is moved to the bin corresponding to the current mask length $m$ if its SSRC matches the key under the masking operation and otherwise is discarded.

To obtain the estimate of the number of session members $L$, the following formula is used:
$L = \sum_{i=0}^{i=31} B(i)2^i$

Where $B(i)$ is the number of SSRC identifiers in bin $i$. Note that this formula counts senders only once since they are all represented in bin 0, but multiplies the sampled count of non-senders (receivers) by the sampling factor.

As participants leave the session by sending a BYE or being timed out, their entries are removed from the table and the number of entries in the table may become too small to provide a reasonable statistical estimate. When this occurs, it is necessary to decrease the number of bits in the mask so that additional SSRC identifiers will be kept. It is recommended that the mask be decreased by one when:

$L/2^m < B/4$

When the mask size is reduced from $m$ to $m-1$, all the SSRC identifiers remain in their current bins. Thus the estimate of the number of session members is not immediately affected by the change in mask size. When a packet arrives from an SSRC that is currently in some bin $x$ where $x < m$, that SSRC identifier is moved from bin $x$ to bin $m$, reducing the estimate. When a BYE packet is received from a participant or the participant is timed out, and the participant's SSRC exists in the membership table, that SSRC identifier is removed from its bin. Thus the contributions from higher bins fade away as bin $m$ acquires a more complete list of the SSRC identifiers that will now be kept because of the reduced mask.

### 6.3 RTCP Packet Send and Receive Rules

The rules for how to send, and what to do when receiving an RTCP packet are outlined here. To execute these rules, a session participant must maintain several pieces of state:

**tp:** the last time an RTCP packet was transmitted;

**tc:** the current time;

**tn:** the next scheduled transmission time of an RTCP packet;

**pmembers:** the estimated number of session members at time tp

**members:** the most current estimate for the number of session members;

**senders:** the most current estimate for the number of senders in the session;

**rtcp_bw:** The target RTCP bandwidth, i.e., the total bandwidth that will be used for RTCP packets by all members of this session, in octets per second. This should be 5% of the "session bandwidth" parameter supplied to the application at startup.

**we_sent:** Flag that is true if the application has sent data since the 2nd previous RTCP report was transmitted.

**avg_rtcp_size:** The average compound RTCP packet size, in octets, over all RTCP packets sent and received by this participant.

**initial:** Flag that is true if the application has not yet sent an RTCP packet.

Many of these rules make use of the "calculated interval" between packet transmissions. This interval is described in the following section.

### 6.3.1 Computing the RTCP transmission interval

To maintain scalability, the average interval between packets from a session participant should scale with the group size. This interval is called the calculated interval. It is obtained by combining a number of the pieces of state described above. The calculated interval T is then determined as follows:

1. If there are any senders (senders > 0) in the session, but the number of senders is less than 25% of the membership (members), the interval depends on whether the participant is a sender or not (based on the value of we_sent). If the participant is a sender (we_sent true), the constant C is set to the average RTCP packet size (avg_rtcp_size) divided by 25% of the RTCP bandwidth (rtcp_bw), and the constant n is set to the number of senders. If we_sent is not true, the constant C is set to the average RTCP packet size divided by 75% of the RTCP bandwidth. The constant n is set to the number of receivers (members − senders). If the number of senders is greater than 25%, senders and receivers are treated together. The constant C is set to the total RTCP bandwidth and n is set to the total number of members.
2. If the participant has not yet sent an RTCP packet (the variable initial is true), the constant Tmin is set to 2.5 seconds, else it is set to 5 seconds.
3. The deterministic calculated interval Td is set to max(Tmin, n*C).
4. The calculated interval T is set to a number uniformly distributed between 0.5 and 1.5 times the deterministic calculated interval.

This procedure results in an interval which is random, but which, on average, gives 25% of the RTCP bandwidth to senders, and 75% to receivers.

### 6.3.2 Initialization

Upon joining the session, the participant initializes tp to 0, tc to 0, senders to 0, pmembers to 1, members to 1, we_sent to false, rtcp_bw to 5% of the session bandwidth, initial to true, and avg_rtcp_size to the size of the very first packet constructed by the application. The calculated interval T is then computed, and the first packet is scheduled for time tn = T. This means that a transmission timer is set which expires at time T. Note that an application MAY use any desired approach for implementing this timer.

The participant adds their own SSRC to the member table.

### 6.3.3 Receiving an RTP or non-BYE RTCP packet

When an RTP or RTCP packet is received from a participant whose SSRC is not in the member table, the SSRC is added to the table, and the value for members is updated.

When an RTP packet is received from a participant whose SSRC is not in the sender table, the SSRC is added to the table, and the value for senders is updated.

For each compound RTCP packet received, the value of avg_rtcp_size is updated: avg_rtcp_size = (1/16)*packet_size + (15/16)* avg_rtcp_size, where packet_size is the size of the RTCP packet just received.

### 6.3.4 Receiving an RTCP BYE packet

Except as described in Section 6.3.7 for the case when an RTCP BYE is to be transmitted, if the received packet is an RTCP BYE packet, the SSRC is checked against the member table. If present, the entry is removed from the table, and the value for members is updated. The SSRC is then checked against the sender table. If present, the entry is removed from the table, and the value for senders is updated.

Furthermore, to make the transmission rate of RTCP packets more adaptive to changes in group membership, the following "reverse reconsideration" algorithm SHOULD be executed when a BYE packet is received:

- The value for tn is updated according to the following formula: tn = tc + (members/pmembers)(tn - tc).
- The value for tp is updated according the following formula: tp = tc - (members/pmembers)(tc - tp).
- The next RTCP packet is rescheduled for transmission at time tn, which is now earlier.
- The value of pmembers is set equal to members.

This algorithm does not prevent the group size estimate from incorrectly dropping to zero for a short time when most participants of a large session leave at once but some remain. The algorithm does make the estimate return to the correct value more rapidly. This situation is unusual enough and the consequences are sufficiently harmless that this problem is deemed only a secondary concern.

### 6.3.5 Timing Out an SSRC

At occassional intervals, the participant MUST check to see if any of the other participants time out. To do this, the participant computes the deterministic calculated interval (without the randomization factor) Td. Any other session member who has not sent a packet since time tc - MTd (M is the timeout multiplier, and defaults to 5) is timed out. This means that their SSRC is removed from the member list, and members is updated. A similar check is performed on the sender list. Any member on the sender list who has not sent an RTP packet since time tc - 2T (within the last two RTCP report intervals) is removed from the sender list, and senders is updated.

If any members time out, the reverse reconsideration algorithm described in Section 6.3.4 SHOULD be performed.

The participant MUST perform this check at least once per RTCP transmission interval.

### 6.3.6 Expiration of transmission timer

When the packet transmission timer expires, the participant performs one of the following operations:

Option A ("conditional reconsideration"):

- If members is less than or equal to pmembers, an RTCP packet is transmitted. The transmission interval T, including the randomization factor, is computed. pmembers is set to members, tp is set to tc, and tn is set to tc + T. The transmission timer is set to expire again at time tn.
- If members is greater than pmembers, the transmission interval T, including the randomization factor, is computed. If tp + T is less than or equal to tc, an RTCP packet is transmitted. pmembers is set to members, tp is set to tc, and tn is set to tc + T. The transmission timer is set to expire again at time tn. If tp + T is greater than tc, pmembers is set to members, and tn is set to tp + T. No RTCP packet is transmitted. The transmission timer is set to expire at time tn.

Option B ("unconditional reconsideration"):

- The transmission interval T is computed, including the randomization factor and a factor $e - 3/2 = 1.21828$ times the rtcp_bw to compensate for the fact that the unconditional reconsideration algorithm converges to a value below the intended average.
- If tp + T is less than or equal to tc, an RTCP packet is transmitted. tp is set to tc, and tn is set to tc + T. The transmission timer is set to expire again at time tn. If tp + T is greater than tc, pmembers is set to members, and tn is set to tp + T. No RTCP packet is transmitted. The transmission timer is set to expire at time tn.

Option C ("hybrid reconsideration"):

- Option B is executed for the first RTCP packet.
- Option A is executed for all subsequent packets.

Implementationss SHOULD use Option B. Implementations MAY use options C and A. Option B provides the best protection against RTCP packet floods in the event of simultaneous joins or when network partitions heal.

If an RTCP packet is transmitted (using any of the above options), the value of initial is set to FALSE. Furthermore, the value of avg_rtcp_size is updated: avg_rtcp_size = (1/16)*packet_size + (15/16)* avg_rtcp_size, where packet_size is the size of the RTCP packet just transmitted.

### 6.3.7 Transmitting a BYE packet

When a participant wishes to leave a session, a BYE packet is transmitted to inform the other participants of the event. In order to avoid a flood of BYE packets when many participants leave the system, a participant MUST execute the following algorithm if the number of members is more than 50 when the participant chooses to leave. This algorithm usurps the normal role of the members variable to count BYE packets instead:

- When the participant decides to leave the system, tp is reset to tc, the current time, members and pmembers are initialized to 1, initial is set to 1, we_sent is set to 0, senders is set to 0, and avg_rtcp_size is set to the size of the BYE packet. The calculated interval T is computed. The BYE packet is then scheduled for time tn = tc + T.
- Every time a BYE packet from another participant is received, members is incremented by 1 regardless of whether that participant exists in the member table or not, and when SSRC sampling is in use, regardless of whether the BYE SSRC matches the key or not. members is NOT incremented when other RTCP packets or RTP packets are received, but only for BYE packets.
- Transmission of the BYE packet then follows the rules for transmitting a regular RTCP packet, as above. Option B SHOULD be used.

This allows BYE packets to be sent right away, yet controls their total bandwidth usage. In the worst case, this could cause RTCP control packets to use twice the bandwidth as normal (10%) — 5% for non BYE RTCP packets and 5% for BYE.

A participant that does not want to wait for the above mechanism to allow transmission of a BYE packet MAY leave the group without sending a BYE at all. That participant will eventually be timed out by the other group members.

If the group size estimate members is less than 50 when the participant decides to leave, the participant MAY send a BYE packet immediately. Alternatively, the participant MAY choose to execute the above BYE backoff algorithm.

In either case, a participant which never sent an RTP or RTCP packet MUST NOT send a BYE packet when they leave the group.

#### 6.3.8 Updating we_sent

The variable we_sent contains true if the participant has sent an RTP packet recently, false otherwise. This determination is made by using the same mechanisms for managing the senders table and sending SR packets. If the participant sends an RTP packet when we_sent is false, it adds itself to the sender table and sets we_sent to true. Every time another RTP packet is sent, the time of transmission of that packet is maintained in the table. The normal sender timeout algorithm is then applied to the participant — if an RTP packet has not been transmitted since time tc - 2T, the participant removes itself from the sender table, decrements the sender count, and sets we_sent to false.

#### 6.3.9 Allocation of source description bandwidth

This specification defines several source description (SDES) items in addition to the mandatory CNAME item, such as NAME (personal name) and EMAIL (email address). It also provides a means to define new application-specific RTCP packet types. Applications should exercise caution in allocating control bandwidth to this additional information because it will slow down the rate at which reception reports and CNAME are sent, thus impairing the performance of the protocol. It is recommended that no more than 20% of the RTCP bandwidth allocated to a single participant be used to carry the additional information. Furthermore, it is *not* intended that all SDES items should be included in every application. Those that are included should be assigned a fraction of the bandwidth according to their utility. Rather than estimate these fractions dynamically, it is recommended that the percentages be translated statically into report interval counts based on the typical length of an item.

For example, an application may be designed to send only CNAME, NAME and EMAIL and not any others. NAME might be given much higher priority than EMAIL because the NAME would be displayed continuously in the application's user interface, whereas EMAIL would be displayed only when requested. At every RTCP interval, an RR packet and an SDES packet with the CNAME item would be sent. For a small session operating at the minimum interval, that would be every 5 seconds on the average. Every third interval (15 seconds), one extra item would be included in the SDES packet. Seven out of eight times this would be the NAME item, and every eighth time (2 minutes) it would be the EMAIL item.

When multiple applications operate in concert using cross-application binding through a common CNAME for each participant, for example in a multimedia conference composed of an RTP session for each medium, the additional SDES information might be sent in only one RTP session. The other sessions would carry only the CNAME item. In particular, this approach should be applied to the multiple sessions of a layered encoding scheme (see Section 2.4).

### 6.4 Sender and Receiver Reports

RTP receivers provide reception quality feedback using RTCP report packets which may take one of two forms depending upon whether or not the receiver is also a sender. The only difference between the sender report (SR) and receiver report (RR) forms, besides the packet type code, is that the sender report includes a 20-byte sender information section for use by active senders. The SR is issued if a site has sent any data packets during the interval since issuing the last report or the previous one, otherwise the RR is issued.

Both the SR and RR forms include zero or more reception report blocks, one for each of the synchronization sources from which this receiver 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. Since a maximum of 31 reception report blocks will fit in an SR or RR packet, additional RR packets may be stacked after the initial SR or RR packet as needed to contain the reception reports for all sources heard during the interval since the last report.

The next sections define the formats of the two reports, how they may be extended in a profile-specific manner if an application requires additional feedback information, and how the reports may be used. Details of reception reporting by translators and mixers is given in Section 7.

### 6.4.1 SR: Sender report RTCP packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    RC   |   PT=SR=200   |             length            | header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SSRC of sender                        |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|              NTP timestamp, most significant word             | sender
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ info
|             NTP timestamp, least significant word             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         RTP timestamp                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     sender's packet count                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      sender's octet count                     |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_1 (SSRC of first source)                 | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
| fraction lost |       cumulative number of packets lost       |   1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           extended highest sequence number received           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      interarrival jitter                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         last SR (LSR)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   delay since last SR (DLSR)                  |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_2 (SSRC of second source)                | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
:                               ...                             :   2
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                  profile-specific extensions                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The sender report packet consists of three sections, possibly followed by a fourth profile-specific extension section if defined. The first section, the header, is 8 octets long. The fields have the following meaning:

**version (V):** 2 bits
: Identifies the version of RTP, which is the same in RTCP packets as in RTP data packets. The version defined by this specification is two (2).

**padding (P):** 1 bit
: If the padding bit is set, this individual RTCP packet contains some additional padding octets at the end which are not part of the control information but are included in the length field. The last octet of the padding is a count of how many padding octets should be ignored, including itself (it will be a multiple of four). Padding may be needed by some encryption algorithms with fixed block sizes. In a compound RTCP packet, padding is only required on one individual packet because the compound

packet is encrypted as a whole for the method in Section 9.1. Thus, padding MUST only be added to the last individual packet, and if padding is added to that packet, the padding bit MUST be set only on that packet. This convention aids the header validity checks described in Appendix A.2 and allows detection of packets from some early implementations that incorrectly set the padding bit on the first individual packet and add padding to the last individual packet.

**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 as an RTCP SR packet.

**length:** 16 bits
The length of this RTCP packet in 32-bit words minus one, including the header and any padding. (The offset of one makes zero a valid length and avoids a possible infinite loop in scanning a compound RTCP packet, while counting 32-bit words avoids a validity check for a multiple of 4.)

**SSRC:** 32 bits
The synchronization source identifier for the originator of this SR packet.

The second section, the sender information, is 20 octets long and is present in every sender report packet. It summarizes the data transmissions from this sender. The fields have the following meaning:

**NTP timestamp:** 64 bits
Indicates the wallclock time (see Section 4) when this report was sent so that it may be used in combination with timestamps returned in reception reports from other receivers to measure round-trip propagation to those receivers. Receivers should expect that the measurement accuracy of the timestamp may be limited to far less than the resolution of the NTP timestamp. The measurement uncertainty of the timestamp is not indicated as it may not be known. On a system that has no notion of wallclock time but does have some system-specific clock such as "system uptime", a sender MAY use that clock as a reference to calculate relative NTP timestamps. It is important to choose a commonly used clock so that if separate implementations are used to produce the individual streams of a multimedia session, all implementations will use the same clock. These relative NTP timestamps are assumed to have a reference time less than 68 years in the past, so the high bit will be zero to serve as an indication of relative timestamps. A sender that has no notion of wallclock or elapsed time may set the NTP timestamp to zero.

**RTP timestamp:** 32 bits
Corresponds to the same time as the NTP timestamp (above), 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, and may be used by media-independent receivers to estimate the nominal RTP clock frequency. Note that in most cases this timestamp will not be equal to the RTP timestamp in any adjacent data packet. Rather, it is calculated from the corresponding NTP timestamp using the relationship between the RTP timestamp counter and real time as maintained by periodically checking the wallclock time at a sampling instant.

**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 SR packet 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 (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission up until the time this SR packet 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 zero or more reception report blocks depending on the number of other sources heard by this sender since the last report. Each reception report block conveys statistics on the reception of RTP packets from a single synchronization source. Receivers do not carry over statistics when a source changes its SSRC identifier due to a collision. These statistics are:

**SSRC_n (source identifier):** 32 bits
The SSRC identifier of the source to which the information in this reception report block pertains.

**fraction lost:** 8 bits
The fraction of RTP data packets from source SSRC_n lost since the previous SR or RR packet was sent, expressed as a fixed point number with the binary point at the left edge of the field. (That is equivalent to taking the integer part after multiplying the loss fraction by 256.) This fraction is defined to be the number of packets lost divided by the number of packets expected, as defined in the next paragraph. An implementation is shown in Appendix A.3. If the loss is negative due to duplicates, the fraction lost is set to zero. Note that a receiver cannot tell whether any packets were lost after the last one received, and that there will be no reception report block issued for a source if all packets from that source sent during the last reporting interval have been lost.

**cumulative number of packets lost:** 24 bits
The total number of RTP data packets from source SSRC_n that have been lost since the beginning of reception. This number is defined to be the number of packets expected less the number of packets actually received, where the number of packets received includes any which are late or duplicates. 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 extended last sequence number received, as defined next, less the initial sequence number received. This may be calculated as shown in Appendix A.3.

**extended highest sequence number received:** 32 bits
The low 16 bits contain the highest sequence number received in an RTP data packet from source SSRC_n, and the most significant 16 bits extend that sequence number with the corresponding count of sequence number cycles, which may be maintained according to the algorithm in Appendix A.1. Note that different receivers within the same session will generate different extensions to the sequence number if their start times differ significantly.

**interarrival jitter:** 32 bits
An estimate of the statistical variance of the RTP data packet interarrival time, measured in timestamp units and expressed as an unsigned integer. The interarrival jitter $J$ is defined to be the mean deviation (smoothed absolute value) of the difference $D$ in packet spacing at the receiver compared to the sender for a pair of packets. As shown in the equation below, this is equivalent to the difference in the "relative transit time" for the two packets; the relative transit time is the difference between a packet's RTP timestamp and the receiver's clock at the time of arrival, measured in the same units.

If $S_i$ is the RTP timestamp from packet $i$, and $R_i$ is the time of arrival in RTP timestamp units for packet $i$, then for two packets $i$ and $j$, $D$ may be expressed as

$$D(i,j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i)$$

The interarrival jitter is calculated continuously as each data packet $i$ is received from source SSRC_n, using this difference $D$ for that packet and the previous packet $i-1$ in order of arrival (not necessarily in sequence), according to the formula

$$J_i = J_{i-1} + (|D(i-1,i)| - J_{i-1})/16$$

Whenever a reception report is issued, the current value of $J$ is sampled.

The jitter calculation is prescribed here to allow profile-independent monitors to make valid interpretations of reports coming from different implementations. This algorithm is the optimal first-order estimator and the gain parameter $1/16$ gives a good noise reduction ratio while maintaining a reasonable rate of convergence [12, Section 11.5-11.12, Fig. 11.6]. A sample implementation is shown in Appendix A.8.

**last SR timestamp (LSR):** 32 bits
The middle 32 bits out of 64 in the NTP timestamp (as explained in Section 4) received as part of the most recent RTCP sender report (SR) packet from source SSRC_n. If no SR has been received yet, the field is set to zero.

**delay since last SR (DLSR):** 32 bits
The delay, expressed in units of $1/65536$ seconds, between receiving the last SR packet from source SSRC_n and sending this reception report block. If no SR packet has been received yet from SSRC_n, the DLSR field is set to zero.

Let SSRC_r denote the receiver issuing this receiver report. Source SSRC_n can compute the round propagation delay to SSRC_r by recording the time $A$ when this reception report block is received. It calculates the total round-trip time $A-$LSR using the last SR timestamp (LSR) field, and then subtracting this field to leave the round-trip propagation delay as ($A-$ LSR $-$ DLSR). This is illustrated in Fig. 2.

This may be used as an approximate measure of distance to cluster receivers, although some links have very asymmetric delays.

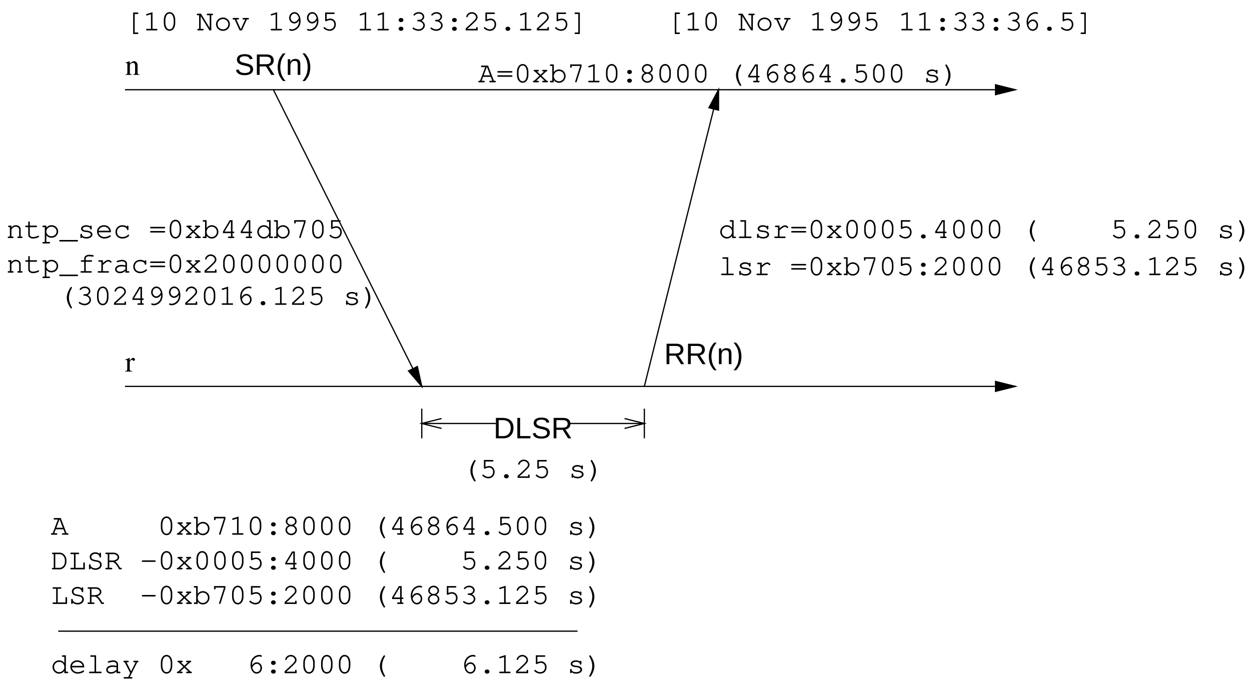


Figure 2: Example for round-trip time computation

#### 6.4.2 RR: Receiver report RTCP packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    RC   |   PT=RR=201   |             length            | header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     SSRC of packet sender                     |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_1 (SSRC of first source)                 | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
| fraction lost |       cumulative number of packets lost       |   1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           extended highest sequence number received           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      interarrival jitter                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         last SR (LSR)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   delay since last SR (DLSR)                  |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_2 (SSRC of second source)                | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
:                               ...                             :   2
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                  profile-specific extensions                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the receiver report (RR) packet is the same as that of the SR packet except that the packet type field contains the constant 201 and the five words of sender information are omitted (these are the NTP and RTP timestamps and sender's packet and octet counts). The remaining fields have the same meaning as for the SR packet.

An empty RR packet (RC = 0) is put at the head of a compound RTCP packet when there is no data transmission or reception to report.

#### 6.4.3 Extending the sender and receiver reports

A profile SHOULD define profile-specific extensions to the sender report and receiver report if there is additional information that needs to be reported regularly about the sender or receivers. This method SHOULD be used in preference to defining another RTCP packet type because it requires less overhead:

- fewer octets in the packet (no RTCP header or SSRC field);
- simpler and faster parsing because applications running under that profile would be programmed to always expect the extension fields in the directly accessible location after the reception reports.

The extension is a fourth section in the sender- or receiver-report packet which comes at the end after the reception report blocks, if any. If additional sender information is required, then for sender reports it should be included first in the extension section, but for receiver reports it would not be present. If information about receivers is to be included, that data may be structured as an array of blocks parallel to the existing array of reception report blocks; that is, the number of blocks would be indicated by the RC field.

#### 6.4.4 Analyzing sender and receiver reports

It is expected that reception quality feedback will be useful not only for the sender but also for other receivers and third-party monitors. The sender may modify its transmissions based on the feedback; receivers can determine whether problems are local, regional or global; network managers may use profile-independent monitors that receive only the RTCP packets and not the corresponding RTP data packets to evaluate the performance of their networks for multicast distribution.

Cumulative counts are used in both the sender information and receiver report blocks so that differences may be calculated between any two reports to make measurements over both short and long time periods, and to provide resilience against the loss of a report. The difference between the last two reports received can be used to estimate the recent quality of the distribution. The NTP timestamp is included so that rates may be calculated from these differences over the interval between two reports. Since that timestamp is independent of the clock rate for the data encoding, it is possible to implement encoding- and profile-independent quality monitors.

An example calculation is the packet loss rate over the interval between two reception reports. The difference in the cumulative number of packets lost gives the number lost during that interval. The difference in the extended last sequence numbers received gives the number of packets expected during the interval. The ratio of these two is the packet loss fraction over the interval. This ratio should equal the fraction lost field if the two reports are consecutive, but otherwise not. The loss rate per second can be obtained by dividing the loss fraction by the difference in NTP timestamps, expressed in seconds. The number of packets received is the number of packets expected minus the number lost. The number of packets expected may also be used to judge the statistical validity of any loss estimates. For example, 1 out of 5 packets lost has a lower significance than 200 out of 1000.

From the sender information, a third-party monitor can calculate the average payload data rate and the average packet rate over an interval without receiving the data. Taking the ratio of the two gives the average payload size. If it can be assumed that packet loss is independent of packet size, then the number of packets received by a particular receiver times the average payload size (or the corresponding packet size) gives the apparent throughput available to that receiver.

In addition to the cumulative counts which allow long-term packet loss measurements using differences between reports, the fraction lost field provides a short-term measurement from a single report. This becomes more important as the size of a session scales up enough that reception state information might not be kept for all receivers or the interval between reports becomes long enough that only one report might have been received from a particular receiver.

The interarrival jitter field provides a second short-term measure of network congestion. Packet loss tracks persistent congestion while the jitter measure tracks transient congestion. The jitter measure may indicate congestion before it leads to packet loss. Since the interarrival jitter field is only a snapshot of the jitter at the time of a report, it may be necessary to analyze a number of reports from one receiver over time or from multiple receivers, e.g., within a single network.

## 6.5 SDES: Source description RTCP packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    SC   |  PT=SDES=202  |             length            | header
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                          SSRC/CSRC_1                          | chunk
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+   1
|                           SDES items                          |
|                              ...                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                          SSRC/CSRC_2                          | chunk
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+   2
|                           SDES items                          |
|                              ...                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The SDES packet is a three-level structure composed of a header and zero or more chunks, each of of which is composed of items describing the source identified in that chunk. The items are described individually in subsequent sections.

**version (V), padding (P), length:**
As described for the SR packet (see Section 6.4.1).

**packet type (PT):** 8 bits
Contains the constant 202 to identify this as an RTCP SDES packet.

**source count (SC):** 5 bits
The number of SSRC/CSRC chunks contained in this SDES packet. A value of zero is valid but useless.

Each chunk consists of an SSRC/CSRC identifier followed by a list of zero or more items, which carry information about the SSRC/CSRC. Each chunk starts on a 32-bit boundary. Each item consists of an 8-bit type field, an 8-bit octet count describing the length of the text (thus, not including this two-octet header), and the text itself. Note that the text can be no longer than 255 octets, but this is consistent with the need to limit RTCP bandwidth consumption.

The text is encoded according to the UTF-8 encoding specified in RFC 2279 [13]. US-ASCII is a subset of this encoding and requires no additional encoding. The presence of multi-octet encodings is indicated by setting the most significant bit of a character to a value of one.

Items are contiguous, i.e., items are not individually padded to a 32-bit boundary. Text is not null terminated because some multi-octet encodings include null octets. The list of items in each chunk is terminated by one or more null octets, the first of which is interpreted as an item type of zero to denote the end of the list. No length octet follows the null item type octet, but additional null octets are included if needed to pad until the next 32-bit boundary. Note that this padding is separate from that indicated by the P bit in the RTCP header. A chunk with zero items (four null octets) is valid but useless.

End systems send one SDES packet containing their own source identifier (the same as the SSRC in the fixed RTP header). A mixer sends one SDES packet containing a chunk for each contributing source from which it is receiving SDES information, or multiple complete SDES packets in the format above if there are more than 31 such sources (see Section 7).

The SDES items currently defined are described in the next sections. Only the CNAME item is mandatory. Some items shown here may be useful only for particular profiles, but the item types are all assigned from one common space to promote shared use and to simplify profile-independent applications. Additional items may be defined in a profile by registering the type numbers with IANA.

### 6.5.1 CNAME: Canonical end-point identifier SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    CNAME=1    |     length    | user and domain name        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The CNAME identifier has the following properties:

- Because the randomly allocated SSRC identifier may change if a conflict is discovered or if a program is restarted, the CNAME item is required to provide the binding from the SSRC identifier to an identifier for the source that remains constant.
- Like the SSRC identifier, the CNAME identifier should also be unique among all participants within one RTP session.
- To provide a binding across multiple media tools used by one participant in a set of related RTP sessions, the CNAME should be fixed for that participant.
- To facilitate third-party monitoring, the CNAME should be suitable for either a program or a person to locate the source.

Therefore, the CNAME should be derived algorithmically and not entered manually, when possible. To meet these requirements, the following format should be used unless a profile specifies an alternate syntax or semantics. The CNAME item should have the format "user@host", or "host" if a user name is not available as on single-user systems. For both formats, "host" is either the fully qualified domain name of the host from which the real-time data originates, formatted according to the rules specified in RFC 1034 [14], RFC 1035 [15] and Section 2.1 of RFC 1123 [16]; or the standard ASCII representation of the host's numeric address on the interface used for the RTP communication. For example, the standard ASCII representation of an IP Version 4 address is "dotted decimal", also known as dotted quad. Other address types are expected to have ASCII representations that are mutually unique. The fully qualified domain name is more convenient for a human observer and may avoid the need to send a NAME item in addition, but it may be difficult or impossible to obtain reliably in some operating environments. Applications that may be run in such environments should use the ASCII representation of the address instead.

Examples are "doe@sleepy.megacorp.com" or "doe@192.0.2.89" for a multi-user system. On a system with no user name, examples would be "sleepy.megacorp.com" or "192.0.2.89".

The user name should be in a form that a program such as "finger" or "talk" could use, i.e., it typically is the login name rather than the personal name. The host name is not necessarily identical to the one in the participant's electronic mail address.

This syntax will not provide unique identifiers for each source if an application permits a user to generate multiple sources from one host. Such an application would have to rely on the SSRC to further identify the source, or the profile for that application would have to specify additional syntax for the CNAME identifier.

If each application creates its CNAME independently, the resulting CNAMEs may not be identical as would be required to provide a binding across multiple media tools belonging to one participant in a set of related RTP sessions. If cross-media binding is required, it may be necessary for the CNAME of each tool to be externally configured with the same value by a coordination tool.

Application writers should be aware that private network address assignments such as the Net-10 assignment proposed in RFC 1597 [17] may create network addresses that are not globally unique. This would lead to non-unique CNAMEs if hosts with private addresses and no direct IP connectivity to the public Internet have their RTP packets forwarded to the public Internet through an RTP-level translator. (See also RFC 1627 [18].) To handle this case, applications may provide a means to configure a unique CNAME, but the burden is on the translator to translate CNAMEs from private addresses to public addresses if necessary to keep private addresses from being exposed.

### 6.5.2 NAME: User name SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     NAME=2    |     length    | common name of source        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This is the real name used to describe the source, e.g., "John Doe, Bit Recycler, Megacorp". It may be in any form desired by the user. For applications such as conferencing, this form of name may be the most desirable for display in participant lists, and therefore might be sent most frequently of those items other than CNAME. Profiles may establish such priorities. The NAME value is expected to remain constant at least for the duration of a session. It should not be relied upon to be unique among all participants in the session.

#### 6.5.3 EMAIL: Electronic mail address SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    EMAIL=3    |     length    | email address of source     ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The email address is formatted according to RFC 822 [19], for example, "John.Doe@megacorp.com". The EMAIL value is expected to remain constant for the duration of a session.

#### 6.5.4 PHONE: Phone number SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    PHONE=4    |     length    | phone number of source      ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The phone number should be formatted with the plus sign replacing the international access code. For example, "+1 908 555 1212" for a number in the United States.

#### 6.5.5 LOC: Geographic user location SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     LOC=5     |     length    | geographic location of site ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Depending on the application, different degrees of detail are appropriate for this item. For conference applications, a string like "Murray Hill, New Jersey" may be sufficient, while, for an active badge system, strings like "Room 2A244, AT&T BL MH" might be appropriate. The degree of detail is left to the implementation and/or user, but format and content may be prescribed by a profile. The LOC value is expected to remain constant for the duration of a session, except for mobile hosts.

#### 6.5.6 TOOL: Application or tool name SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     TOOL=6    |     length    | name/version of source appl. ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

A string giving the name and possibly version of the application generating the stream, e.g., "videotool 1.2". This information may be useful for debugging purposes and is similar to the Mailer or Mail-System-

Version SMTP headers. The TOOL value is expected to remain constant for the duration of the session.

### 6.5.7 NOTE: Notice/status SDES item

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     NOTE=7    |     length    | note about the source        ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following semantics are suggested for this item, but these or other semantics may be explicitly defined by a profile. The NOTE item is intended for transient messages describing the current state of the source, e.g., “on the phone, can’t talk”. Or, during a seminar, this item might be used to convey the title of the talk. It should be used only to carry exceptional information and should not be included routinely by all participants because this would slow down the rate at which reception reports and CNAME are sent, thus impairing the performance of the protocol. In particular, it should not be included as an item in a user’s configuration file nor automatically generated as in a quote-of-the-day.

Since the NOTE item may be important to display while it is active, the rate at which other non-CNAME items such as NAME are transmitted might be reduced so that the NOTE item can take that part of the RTCP bandwidth. When the transient message becomes inactive, the NOTE item should continue to be transmitted a few times at the same repetition rate but with a string of length zero to signal the receivers. However, receivers should also consider the NOTE item inactive if it is not received for a small multiple of the repetition rate, or perhaps 20-30 RTCP intervals.

### 6.5.8 PRIV: Private extensions SDES item

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     PRIV=8    |     length    | prefix length | prefix string...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
...              |                  value string               ...
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This item is used to define experimental or application-specific SDES extensions. The item contains a prefix consisting of a length-string pair, followed by the value string filling the remainder of the item and carrying the desired information. The prefix length field is 8 bits long. The prefix string is a name chosen by the person defining the PRIV item to be unique with respect to other PRIV items this application might receive. The application creator might choose to use the application name plus an additional subtype identification if needed. Alternatively, it is recommended that others choose a name based on the entity they represent, then coordinate the use of the name within that entity.

Note that the prefix consumes some space within the item’s total length of 255 octets, so the prefix should be kept as short as possible. This facility and the constrained RTCP bandwidth should not be overloaded; it is not intended to satisfy all the control communication requirements of all applications.

SDES PRIV prefixes will not be registered by IANA. If some form of the PRIV item proves to be of general utility, it should instead be assigned a regular SDES item type registered with IANA so that no prefix is required. This simplifies use and increases transmission efficiency.

### 6.6 BYE: Goodbye RTCP packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    SC   |   PT=BYE=203  |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           SSRC/CSRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                              ...                              :
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|     length    |               reason for leaving            ... (opt)
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The BYE packet indicates that one or more sources are no longer active.

**version (V), padding (P), length:**
As described for the SR packet (see Section 6.4.1).

**packet type (PT):** 8 bits
Contains the constant 203 to identify this as an RTCP BYE packet.

**source count (SC):** 5 bits
The number of SSRC/CSRC identifiers included in this BYE packet. A count value of zero is valid, but useless.

The rules for when a BYE packet should be sent are specified in Section 6.3.7.

If a BYE packet is received by a mixer, the mixer forwards the BYE packet with the SSRC/CSRC identifier(s) unchanged. If a mixer shuts down, it should send a BYE packet listing all contributing sources it handles, as well as its own SSRC identifier. Optionally, the BYE packet may include an 8-bit octet count followed by that many octets of text indicating the reason for leaving, e.g., “camera malfunction” or “RTP loop detected”. The string has the same encoding as that described for SDES. If the string fills the packet to the next 32-bit boundary, the string is not null terminated. If not, the BYE packet is padded with null octets to the next 32-bit boundary. This padding is separate from that indicated by the P bit in the RTCP header.

### 6.7 APP: Application-defined RTCP packet

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P| subtype |   PT=APP=204  |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           SSRC/CSRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          name (ASCII)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   application-dependent data                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The APP packet is intended for experimental use as new applications and new features are developed, without requiring packet type value registration. APP packets with unrecognized names should be ignored. After testing and if wider use is justified, it is recommended that each APP packet be redefined without the subtype and name fields and registered with the Internet Assigned Numbers Authority using an RTCP packet type.

**version (V), padding (P), length:**
As described for the SR packet (see Section 6.4.1).

**subtype:** 5 bits
May be used as a subtype to allow a set of APP packets to be defined under one unique name, or for any application-dependent data.

**packet type (PT):** 8 bits
Contains the constant 204 to identify this as an RTCP APP packet.

**name:** 4 octets
A name chosen by the person defining the set of APP packets to be unique with respect to other APP packets this application might receive. The application creator might choose to use the application name, and then coordinate the allocation of subtype values to others who want to define new packet types for the application. Alternatively, it is recommended that others choose a name based on the entity they represent, then coordinate the use of the name within that entity. The name is interpreted as a sequence of four ASCII characters, with uppercase and lowercase characters treated as distinct.

**application-dependent data:** variable length
Application-dependent data may or may not appear in an APP packet. It is interpreted by the application and not RTP itself. It must be a multiple of 32 bits long.

# 7 RTP Translators and Mixers

In addition to end systems, RTP supports the notion of "translators" and "mixers", which could be considered as "intermediate systems" at the RTP level. Although this support adds some complexity to the protocol, the need for these functions has been clearly established by experiments with multicast audio and video applications in the Internet. Example uses of translators and mixers given in Section 2.3 stem from the presence of firewalls and low bandwidth connections, both of which are likely to remain.

### 7.1 General Description

An RTP translator/mixer connects two or more transport-level "clouds". Typically, each cloud is defined by a common network and transport protocol (e.g., IP/UDP) plus a multicast address and transport level destination port or a pair of unicast addresses and ports. (Network-level protocol translators, such as IP version 4 to IP version 6, may be present within a cloud invisibly to RTP.) One system may serve as a translator or mixer for a number of RTP sessions, but each is considered a logically separate entity.

In order to avoid creating a loop when a translator or mixer is installed, the following rules must be observed:

- Each of the clouds connected by translators and mixers participating in one RTP session either must be distinct from all the others in at least one of these parameters (protocol, address, port), or must be isolated at the network level from the others.
- A derivative of the first rule is that there must not be multiple translators or mixers connected in parallel unless by some arrangement they partition the set of sources to be forwarded.

Similarly, all RTP end systems that can communicate through one or more RTP translators or mixers share the same SSRC space, that is, the SSRC identifiers must be unique among all these end systems. Section 8.2 describes the collision resolution algorithm by which SSRC identifiers are kept unique and loops are detected.

There may be many varieties of translators and mixers designed for different purposes and applications. Some examples are to add or remove encryption, change the encoding of the data or the underlying protocols, or replicate between a multicast address and one or more unicast addresses. The distinction between translators and mixers is that a translator passes through the data streams from different sources separately, whereas a mixer combines them to form one new stream:

**Translator:** Forwards RTP packets with their SSRC identifier intact; this makes it possible for receivers to identify individual sources even though packets from all the sources pass through the same translator and carry the translator's network source address. Some kinds of translators will pass through the data untouched, but others may change the encoding of the data and thus the RTP data payload type and timestamp. If multiple data packets are re-encoded into one, or vice versa, a translator must assign new sequence numbers to the outgoing packets. Losses in the incoming packet stream may induce corresponding gaps in the outgoing sequence numbers. Receivers cannot detect the presence of a translator unless they know by some other means what payload type or transport address was used by the original source.

**Mixer:** Receives streams of RTP data packets from one or more sources, possibly changes the data format, combines the streams in some manner and then forwards the combined stream. Since the timing among multiple input sources will not generally be synchronized, the mixer will make timing adjustments among the streams and generate its own timing for the combined stream, so it is the synchronization source. Thus, all data packets forwarded by a mixer will be marked with the mixer's own SSRC identifier. In order to preserve the identity of the original sources contributing to the mixed packet, the mixer should insert their SSRC identifiers into the CSRC identifier list following the fixed RTP header of the packet. A mixer that is also itself a contributing source for some packet should explicitly include its own SSRC identifier in the CSRC list for that packet.

For some applications, it may be acceptable for a mixer not to identify sources in the CSRC list. However, this introduces the danger that loops involving those sources could not be detected.

The advantage of a mixer over a translator for applications like audio is that the output bandwidth is limited to that of one source even when multiple sources are active on the input side. This may be important for low-bandwidth links. The disadvantage is that receivers on the output side don't have any control over which sources are passed through or muted, unless some mechanism is implemented for remote control of the mixer. The regeneration of synchronization information by mixers also means that receivers can't do inter-media synchronization of the original streams. A multi-media mixer could do it.

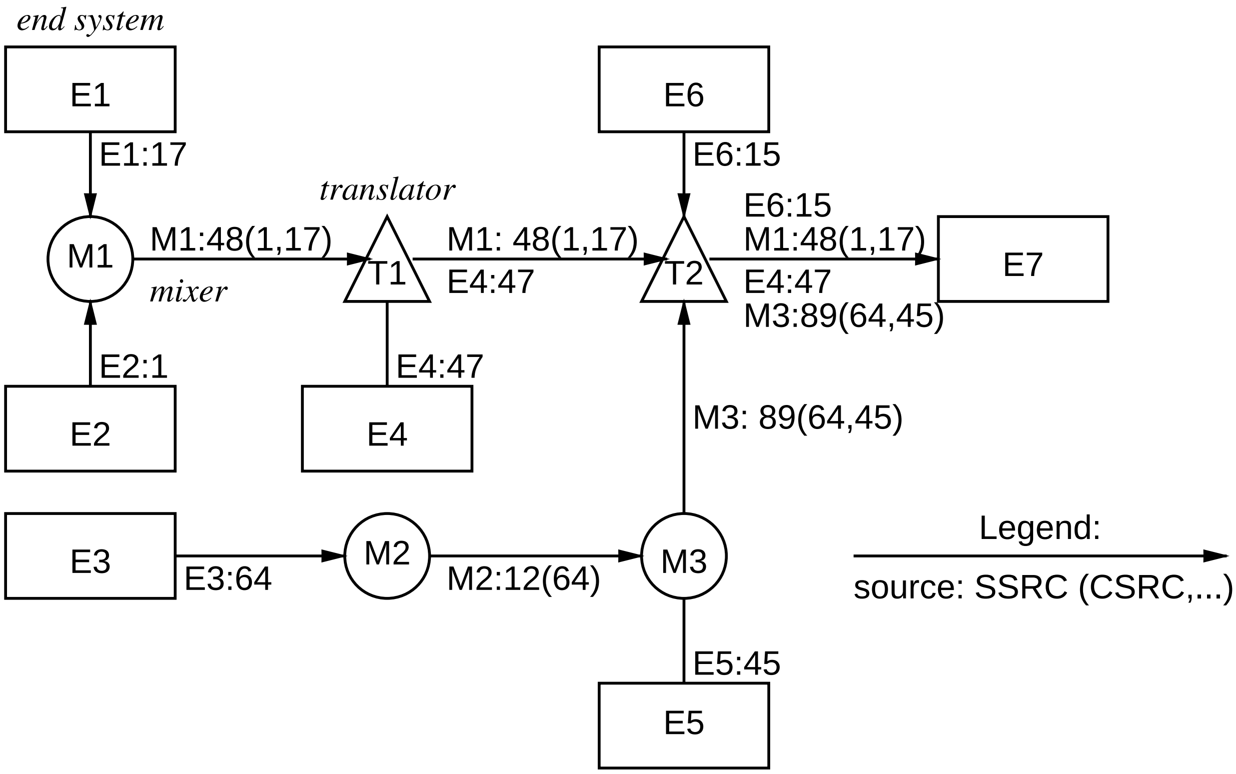


Figure 3: Sample RTP network with end systems, mixers and translators

A collection of mixers and translators is shown in Figure 3 to illustrate their effect on SSRC and CSRC identifiers. In the figure, end systems are shown as rectangles (named E), translators as triangles (named T) and mixers as ovals (named M). The notation "M1: 48(1,17)" designates a packet originating a mixer M1, identified with M1's (random) SSRC value of 48 and two CSRC identifiers, 1 and 17, copied from the SSRC identifiers of packets from E1 and E2.

### 7.2 RTCP Processing in Translators

In addition to forwarding data packets, perhaps modified, translators and mixers must also process RTCP packets. In many cases, they will take apart the compound RTCP packets received from end systems to aggregate SDES information and to modify the SR or RR packets. Retransmission of this information may be triggered by the packet arrival or by the RTCP interval timer of the translator or mixer itself.

A translator that does not modify the data packets, for example one that just replicates between a multicast address and a unicast address, may simply forward RTCP packets unmodified as well. A translator that transforms the payload in some way must make corresponding transformations in the SR and RR information so that it still reflects the characteristics of the data and the reception quality. These translators must not simply forward RTCP packets. In general, a translator should not aggregate SR and RR packets from different sources into one packet since that would reduce the accuracy of the propagation delay measurements based on the LSR and DLSR fields.

**SR sender information:** A translator does not generate its own sender information, but forwards the SR packets received from one cloud to the others. The SSRC is left intact but the sender information must be modified if required by the translation. If a translator changes the data encoding, it must change the "sender's byte count" field. If it also combines several data packets into one output packet, it must change the "sender's packet count" field. If it changes the timestamp frequency, it must change the "RTP timestamp" field in the SR packet.

**SR/RR reception report blocks:** A translator forwards reception reports received from one cloud to the others. Note that these flow in the direction opposite to the data. The SSRC is left intact. If a translator combines several data packets into one output packet, and therefore changes the sequence numbers, it must make the inverse manipulation for the packet loss fields and the "extended last sequence number" field. This may be complex. In the extreme case, there may be no meaningful way to translate the reception reports, so the translator may pass on no reception report at all or a synthetic report based on its own reception. The general rule is to do what makes sense for a particular translation.

A translator does not require an SSRC identifier of its own, but may choose to allocate one for the purpose of sending reports about what it has received. These would be sent to all the connected clouds, each corresponding to the translation of the data stream as sent to that cloud, since reception reports are normally multicast to all participants.

**SDES:** Translators typically forward without change the SDES information they receive from one cloud to the others, but may, for example, decide to filter non-CNAME SDES information if bandwidth is limited. The CNAMEs must be forwarded to allow SSRC identifier collision detection to work. A translator that generates its own RR packets must send SDES CNAME information about itself to the same clouds that it sends those RR packets.

**BYE:** Translators forward BYE packets unchanged. A translator that is about to cease forwarding packets should send a BYE packet to each connected cloud containing all the SSRC identifiers that were previously being forwarded to that cloud, including the translator's own SSRC identifier if it sent reports of its own.

**APP:** Translators forward APP packets unchanged.

### 7.3 RTCP Processing in Mixers

Since a mixer generates a new data stream of its own, it does not pass through SR or RR packets at all and instead generates new information for both sides.

**SR sender information:** A mixer does not pass through sender information from the sources it mixes because the characteristics of the source streams are lost in the mix. As a synchronization source, the mixer generates its own SR packets with sender information about the mixed data stream and sends them in the same direction as the mixed stream.

**SR/RR reception report blocks:** A mixer generates its own reception reports for sources in each cloud and sends them out only to the same cloud. It does not send these reception reports to the other clouds and does not forward reception reports from one cloud to the others because the sources would not be SSRCs there (only CSRCs).

**SDES:** Mixers typically forward without change the SDES information they receive from one cloud to the others, but may, for example, decide to filter non-CNAME SDES information if bandwidth is limited. The CNAMEs must be forwarded to allow SSRC identifier collision detection to work. (An identifier in a CSRC list generated by a mixer might collide with an SSRC identifier generated by an end system.) A mixer must send SDES CNAME information about itself to the same clouds that it sends SR or RR packets.

Since mixers do not forward SR or RR packets, they will typically be extracting SDES packets from a compound RTCP packet. To minimize overhead, chunks from the SDES packets may be aggregated into a single SDES packet which is then stacked on an SR or RR packet originating from the mixer. The RTCP packet rate may be different on each side of the mixer.

A mixer that does not insert CSRC identifiers may also refrain from forwarding SDES CNAMEs. In this case, the SSRC identifier spaces in the two clouds are independent. As mentioned earlier, this mode of operation creates a danger that loops can't be detected.

**BYE:** Mixers need to forward BYE packets. A mixer that is about to cease forwarding packets should send a BYE packet to each connected cloud containing all the SSRC identifiers that were previously being forwarded to that cloud, including the mixer's own SSRC identifier if it sent reports of its own.

**APP:** The treatment of APP packets by mixers is application-specific.

### 7.4 Cascaded Mixers

An RTP session may involve a collection of mixers and translators as shown in Figure 3. If two mixers are cascaded, such as M2 and M3 in the figure, packets received by a mixer may already have been mixed and may include a CSRC list with multiple identifiers. The second mixer should build the CSRC list for the outgoing packet using the CSRC identifiers from already-mixed input packets and the SSRC identifiers from unmixed input packets. This is shown in the output arc from mixer M3 labeled M3:89(64,45) in the figure. As in the case of mixers that are not cascaded, if the resulting CSRC list has more than 15 identifiers, the remainder cannot be included.

## 8 SSRC Identifier Allocation and Use

The SSRC identifier carried in the RTP header and in various fields of RTCP packets is a random 32-bit number that is required to be globally unique within an RTP session. It is crucial that the number be chosen with care in order that participants on the same network or starting at the same time are not likely to choose the same number.

It is *not* sufficient to use the local network address (such as an IPv4 address) for the identifier because the address may not be unique. Since RTP translators and mixers enable interoperation among multiple networks with different address spaces, the allocation patterns for addresses within two spaces might result in a much higher rate of collision than would occur with random allocation. Multiple sources running on one host would also conflict.

It is also not sufficient to obtain an SSRC identifier simply by calling `random()` without carefully initializing the state. An example of how to generate a random identifier is presented in Appendix A.6.

## 8.1 Probability of Collision

Since the identifiers are chosen randomly, it is possible that two or more sources will choose the same number. Collision occurs with the highest probability when all sources are started simultaneously, for example when triggered automatically by some session management event. If $N$ is the number of sources and $L$ the length of the identifier (here, 32 bits), the probability that two sources independently pick the same value can be approximated for large $N$ [20, p. 33] as $1 - e^{-N^2/2^{L+1}}$. For $N = 1000$, the probability is roughly $10^{-4}$.

The typical collision probability is much lower than the worst-case above. When one new source joins an RTP session in which all the other sources already have unique identifiers, the probability of collision is just the fraction of numbers used out of the space. Again, if $N$ is the number of sources and $L$ the length of the identifier, the probability of collision is $N/2^L$. For $N = 1000$, the probability is roughly $2 \cdot 10^{-7}$.

The probability of collision is further reduced by the opportunity for a new source to receive packets from other participants before sending its first packet (either data or control). If the new source keeps track of the other participants (by SSRC identifier), then before transmitting its first packet the new source can verify that its identifier does not conflict with any that have been received, or else choose again.

## 8.2 Collision Resolution and Loop Detection

Although the probability of SSRC identifier collision is low, all RTP implementations must be prepared to detect collisions and take the appropriate actions to resolve them. If a source discovers at any time that another source is using the same SSRC identifier as its own, it must send an RTCP BYE packet for the old identifier and choose another random one. (As explained below, this step is taken only once in case of a loop.) If a receiver discovers that two other sources are colliding, it may keep the packets from one and discard the packets from the other when this can be detected by different source transport addresses or CNAMEs. The two sources are expected to resolve the collision so that the situation doesn't last.

Because the random SSRC identifiers are kept globally unique for each RTP session, they can also be used to detect loops that may be introduced by mixers or translators. A loop causes duplication of data and control information, either unmodified or possibly mixed, as in the following examples:

- A translator may incorrectly forward a packet to the same multicast group from which it has received the packet, either directly or through a chain of translators. In that case, the same packet appears several times, originating from different network sources.
- Two translators incorrectly set up in parallel, i.e., with the same multicast groups on both sides, would both forward packets from one multicast group to the other. Unidirectional translators would produce two copies; bidirectional translators would form a loop.
- A mixer can close a loop by sending to the same transport destination upon which it receives packets, either directly or through another mixer or translator. In this case a source might show up both as an

SSRC on a data packet and a CSRC in a mixed data packet.

A source may discover that its own packets are being looped, or that packets from another source are being looped (a third-party loop).

Both loops and collisions in the random selection of a source identifier result in packets arriving with the same SSRC identifier but a different source transport address, which may be that of the end system originating the packet or an intermediate system. Therefore, if a source changes its source transport address, it must also choose a new SSRC identifier to avoid being interpreted as a looped source. Note that if a translator restarts and consequently changes the source transport address (e.g., changes the UDP source port number) on which it forwards packets, then all those packets will appear to receivers to be looped because the SSRC identifiers are applied by the original source and will not change. This problem may be avoided by keeping the source transport addressed fixed across restarts, but in any case will be resolved after a timeout at the receivers.

Loops or collisions occurring on the far side of a translator or mixer cannot be detected using the source transport address if all copies of the packets go through the translator or mixer, however collisions may still be detected when chunks from two RTCP SDES packets contain the same SSRC identifier but different CNAMEs.

To detect and resolve these conflicts, an RTP implementation must include an algorithm similar to the one described below. It ignores packets from a new source or loop that collide with an established source. It resolves collisions with the participant's own SSRC identifier by sending an RTCP BYE for the old identifier and choosing a new one. However, when the collision was induced by a loop of the participant's own packets, the algorithm will choose a new identifier only once and thereafter ignore packets from the looping source transport address. This is required to avoid a flood of BYE packets.

This algorithm requires keeping a table indexed by the source identifier and containing the source transport addresses from the first RTP packet and first RTCP packet received with that identifier, along with other state for that source. Two source transport addresses are required since, for example, the UDP source port numbers may be different on RTP and RTCP packets. However, it may be assumed that the network address is the same in both source transport addresses.

Each SSRC or CSRC identifier received in an RTP or RTCP packet is looked up in the source identifier table in order to process that data or control information. The source transport address from the packet is compared to the corresponding source transport address in the table to detect a loop or collision if they don't match. For control packets, each element with its own SSRC id, for example an SDES chunk, requires a separate lookup. (The SSRC id in a reception report block is an exception because it identifies a source heard by the reporter, and that SSRC id is unrelated to the source transport adddress of the RTCP packet sent by the reporter.) If the SSRC or CSRC is not found, a new entry is created. These table entries are removed when an RTCP BYE packet is received with the corresponding SSRC id and validated by a matching source transport address, or after no packets have arrived for a relatively long time (see Section 6.2.1).

Note that if two sources on the same host are transmitting with the same source identifier at the time a receiver begins operation, it would be possible that the first RTP packet received came from one of the sources while the first RTCP packet received came from the other. This would cause the wrong RTCP information to be associated with the RTP data, but this situation should be sufficiently rare and harmless that it may be disregarded.

In order to track loops of the participant's own data packets, it is also necessary to keep a separate list of source transport addresses (not identifiers) that have been found to be conflicting. As in the source identifier table, two source transport addresses must be kept to separately track conflicting RTP and RTCP packets. Note that the conflicting address list should be a short, usually empty. Each element in this list stores the source addresses plus the time when the most recent conflicting packet was received. An element may be removed from the list when no conflicting packet has arrived from that source for a time on the order of 10 RTCP report intervals (see Section 6.2).

For the algorithm as shown, it is assumed that the participant's own source identifier and state are included in the source identifier table. The algorithm could be restructured to first make a separate comparison against the participant's own source identifier.

```
IF the SSRC or CSRC identifier is not found in the source
   identifier table:
THEN create a new entry storing the data or control source
     transport address, the SSRC or CSRC id and other state.
     CONTINUE with normal processing.

(identifier is found in the table)

IF the table entry was created on receipt of a control packet
   and this is the first data packet or vice versa:
THEN store the source transport address from this packet.
     CONTINUE with normal processing.
IF the source transport address from the packet matches
   the one saved in the table entry for this identifier:
THEN CONTINUE with normal processing.

(an identifier collision or a loop is indicated)

IF the source identifier is not the participant's own:
THEN IF the source identifier is from an RTCP SDES chunk
        containing a CNAME item that differs from the CNAME
        in the table entry:
     THEN (optionally) count a third-party collision.
     ELSE (optionally) count a third-party loop.
     ABORT processing of data packet or control element.

(a collision or loop of the participant's own packets)

IF the source transport address is found in the list of
   conflicting data or control source transport addresses:
THEN IF the source identifier is not from an RTCP SDES chunk
        containing a CNAME item OR if that CNAME is the
        participant's own:
     THEN (optionally) count occurrence of own traffic looped.
     mark current time in conflicting address list entry.
     ABORT processing of data packet or control element.
log occurrence of a collision.
create a new entry in the conflicting data or control source
   transport address list and mark current time.
```

```
    send an RTCP BYE packet with the old SSRC identifier.
    choose a new identifier.
    create a new entry in the source identifier table with the
       old SSRC plus the source transport address from the data
       or control packet being processed.
    CONTINUE with normal processing.
```

In this algorithm, packets from a newly conflicting source address will be ignored and packets from the original source will be kept. (If the original source was through a mixer and later the same source is received directly, the receiver may be well advised to switch unless other sources in the mix would be lost.) If no packets arrive from the original source for an extended period, the table entry will be timed out and the new source will be able to take over. This might occur if the original source detects the collision and moves to a new source identifier, but in the usual case an RTCP BYE packet will be received from the original source to delete the state without having to wait for a timeout.

When a new SSRC identifier is chosen due to a collision, the candidate identifier should first be looked up in the source identifier table to see if it was already in use by some other source. If so, another candidate should be generated and the process repeated.

A loop of data packets to a multicast destination can cause severe network flooding. All mixers and translators are required to implement a loop detection algorithm like the one here so that they can break loops. This should limit the excess traffic to no more than one duplicate copy of the original traffic, which may allow the session to continue so that the cause of the loop can be found and fixed. However, in extreme cases where a mixer or translator does not properly break the loop and high traffic levels result, it may be necessary for end systems to cease transmitting data or control packets entirely. This decision may depend upon the application. An error condition should be indicated as appropriate. Transmission might be attempted again periodically after a long, random time (on the order of minutes).

### 8.3 Use with Layered Encodings

For layered encodings transmitted on separate RTP sessions (see Section 2.4), a single SSRC identifier space should be used across the sessions of all layers and the core (base) layer should be used for SSRC identifier allocation and collision resolution. When a source discovers that it has collided, it transmits an RTCP BYE message on only the base layer but changes the SSRC identifier to the new value in all layers.

## 9 Security

Lower layer protocols may eventually provide all the security services that may be desired for applications of RTP, including authentication, integrity, and confidentiality. These services have been specified for IP in [21]. Since the initial audio and video applications using RTP needed a confidentiality service before such services were available for the IP layer, the confidentiality service described in the next section was defined for use with RTP and RTCP. That description is included here to codify existing practice. New applications of RTP MAY implement this RTP-specific confidentiality service for backward compatibility, and/or they MAY implement IP layer security services. The overhead on the RTP protocol for this confidentiality service is low, so the penalty will be minimal if this service is obsoleted by lower layer services in the future.

Alternatively, other services, other implementations of services and other algorithms may be defined for RTP in the future if warranted. The selection presented here is meant to simplify implementation of interoperable, secure applications and provide guidance to implementors. No claim is made that the methods presented here are appropriate for a particular security need. A profile may specify which services and algorithms should be offered by applications, and may provide guidance as to their appropriate use.

Key distribution and certificates are outside the scope of this document.

## 9.1 Confidentiality

Confidentiality means that only the intended receiver(s) can decode the received packets; for others, the packet contains no useful information. Confidentiality of the content is achieved by encryption.

When encryption of RTP or RTCP is desired, all the octets that will be encapsulated for transmission in a single lower-layer packet are encrypted as a unit. For RTCP, a 32-bit random number is prepended to the unit before encryption to deter known plaintext attacks. For RTP, no prefix is required because the sequence number and timestamp fields are initialized with random offsets.

For RTCP, it is allowed to split a compound RTCP packet into two lower-layer packets, one to be encrypted and one to be sent in the clear. For example, SDES information might be encrypted while reception reports were sent in the clear to accommodate third-party monitors that are not privy to the encryption key. In this example, depicted in Fig. 4, the SDES information must be appended to an RR packet with no reports (and the encrypted) to satisfy the requirement that all compound RTCP packets begin with an SR or RR packet.

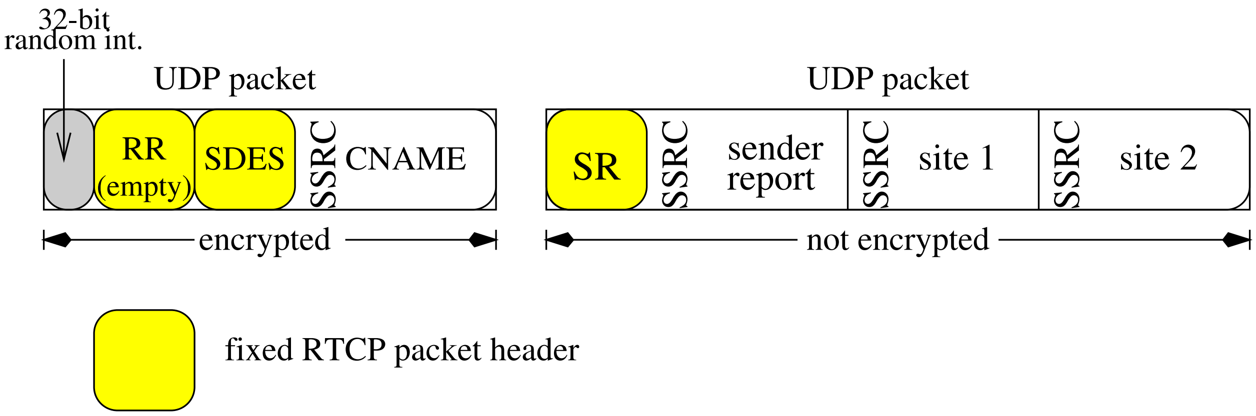


Figure 4: Encrypted and non-encrypted RTCP packets

The presence of encryption and the use of the correct key are confirmed by the receiver through header or payload validity checks. Examples of such validity checks for RTP and RTCP headers are given in Appendices A.1 and A.2.

The default encryption algorithm is the Data Encryption Standard (DES) algorithm in cipher block chaining (CBC) mode, as described in Section 1.1 of RFC 1423 [22], except that padding to a multiple of 8 octets is indicated as described for the P bit in Section 5.1. The initialization vector is zero because random values are supplied in the RTP header or by the random prefix for compound RTCP packets. For details on the use of CBC initialization vectors, see [23]. Implementations that support encryption should always support the DES algorithm in CBC mode as the default to maximize interoperability. This method is chosen because it has been demonstrated to be easy and practical to use in experimental audio and video tools in operation on the Internet. Other encryption algorithms may be specified dynamically for a session by non-RTP means.

As an alternative to encryption at the IP level or at the RTP level as described above, profiles may define additional payload types for encrypted encodings. Those encodings must specify how padding and other aspects of the encryption should be handled. This method allows encrypting only the data while leaving the headers in the clear for applications where that is desired. It may be particularly useful for hardware devices that will handle both decryption and decoding.

### 9.2 Authentication and Message Integrity

Authentication and message integrity services are not defined at the RTP level since these services would not be directly feasible without a key management infrastructure. It is expected that authentication and integrity services will be provided by lower layer protocols.

## 10 RTP over Network and Transport Protocols

This section describes issues specific to carrying RTP packets within particular network and transport protocols. The following rules apply unless superseded by protocol-specific definitions outside this specification.

RTP relies on the underlying protocol(s) to provide demultiplexing of RTP data and RTCP control streams. For UDP and similar protocols, RTP uses an even port number and the corresponding RTCP stream uses the next higher (odd) port number. If an application is supplied with an odd number for use as the RTP port, it should replace this number with the next lower (even) number.

In a unicast session, applications should be prepared to receive RTP data and control on one port pair and send to another.

It is recommended that layered encoding applications (see Section 2.4) use a set of contiguous port numbers. Ports must be distinct because of a widespread deficiency in existing operating systems that prevents use of the same port with multiple multicast addresses, and for unicast, there is only one permissible address. Thus for layer n, the data port is P + 2n, and the control port is P + 2n + 1. When IP multicast is used, the addresses must also be distinct because multicast routing and group membership are managed on an address granularity. However, allocation of contiguous IP multicast addresses cannot be assumed because some groups may require different scopes and may therefore be allocated from different address ranges.

RTP data packets contain no length field or other delineation, therefore RTP relies on the underlying protocol(s) to provide a length indication. The maximum length of RTP packets is limited only by the underlying protocols.

If RTP packets are to be carried in an underlying protocol that provides the abstraction of a continuous octet stream rather than messages (packets), an encapsulation of the RTP packets must be defined to provide a framing mechanism. Framing is also needed if the underlying protocol may contain padding so that the extent of the RTP payload cannot be determined. The framing mechanism is not defined here.

A profile may specify a framing method to be used even when RTP is carried in protocols that do provide framing in order to allow carrying several RTP packets in one lower-layer protocol data unit, such as a UDP packet. Carrying several RTP packets in one network or transport packet reduces header overhead and may simplify synchronization between different streams.

## 11 Summary of Protocol Constants

This section contains a summary listing of the constants defined in this specification.

The RTP payload type (PT) constants are defined in profiles rather than this document. However, the octet of the RTP header which contains the marker bit(s) and payload type must avoid the reserved values 200 and 201 (decimal) to distinguish RTP packets from the RTCP SR and RR packet types for the header validation procedure described in Appendix A.1. For the standard definition of one marker bit and a 7-bit payload type field as shown in this specification, this restriction means that payload types 72 and 73 are reserved.

### 11.1 RTCP packet types

| abbrev. | name | value |
|---|---|---|
| SR | sender report | 200 |
| RR | receiver report | 201 |
| SDES | source description | 202 |
| BYE | goodbye | 203 |
| APP | application-defined | 204 |

These type values were chosen in the range 200-204 for improved header validity checking of RTCP packets compared to RTP packets or other unrelated packets. When the RTCP packet type field is compared to the corresponding octet of the RTP header, this range corresponds to the marker bit being 1 (which it usually is not in data packets) and to the high bit of the standard payload type field being 1 (since the static payload types are typically defined in the low half). This range was also chosen to be some distance numerically from 0 and 255 since all-zeros and all-ones are common data patterns.

Since all compound RTCP packets must begin with SR or RR, these codes were chosen as an even/odd pair to allow the RTCP validity check to test the maximum number of bits with mask and value.

Other constants are assigned by IANA. Experimenters are encouraged to register the numbers they need for experiments, and then unregister those which prove to be unneeded.

### 11.2 SDES types

| abbrev. | name | value |
|---|---|---|
| END | end of SDES list | 0 |
| CNAME | canonical name | 1 |
| NAME | user name | 2 |
| EMAIL | user's electronic mail address | 3 |
| PHONE | user's phone number | 4 |
| LOC | geographic user location | 5 |
| TOOL | name of application or tool | 6 |
| NOTE | notice about the source | 7 |
| PRIV | private extensions | 8 |

Other constants are assigned by IANA. Experimenters are encouraged to register the numbers they need for experiments, and then unregister those which prove to be unneeded.

## 12 RTP Profiles and Payload Format Specifications

A complete specification of RTP for a particular application will require one or more companion documents of two types described here: profiles, and payload format specifications.

RTP may be used for a variety of applications with somewhat differing requirements. The flexibility to adapt to those requirements is provided by allowing multiple choices in the main protocol specification, then selecting the appropriate choices or defining extensions for a particular environment and class of applications in a separate *profile* document. Typically an application will operate under only one profile so there is no explicit indication of which profile is in use. A profile for audio and video applications may be found in the companion RFC 1890 (updated by Internet-Draft `draft-ietf-avt-profile-new`). Profiles are typically titled "RTP Profile for …".

The second type of companion document is a *payload format* specification, which defines how a particular kind of payload data, such as H.261 encoded video, should be carried in RTP. These documents are typically titled "RTP Payload Format for XYZ Audio/Video Encoding". Payload formats may be useful under multiple profiles and may therefore be defined independently of any particular profile. The profile documents are then responsible for assigning a default mapping of that format to a payload type value if needed.

Within this specification, the following items have been identified for possible definition within a profile, but this list is not meant to be exhaustive:

**RTP data header:** The octet in the RTP data header that contains the marker bit and payload type field may be redefined by a profile to suit different requirements, for example with more or fewer marker bits (Section 5.3, p. 13).

**Payload types:** Assuming that a payload type field is included, the profile will usually define a set of payload formats (e.g., media encodings) and a default static mapping of those formats to payload type values. Some of the payload formats may be defined by reference to separate payload format specifications. For each payload type defined, the profile must specify the RTP timestamp clock rate to be used (Section 5.1, p. 12).

**RTP data header additions:** Additional fields may be appended to the fixed RTP data header if some additional functionality is required across the profile's class of applications independent of payload type (Section 5.3, p. 13).

**RTP data header extensions:** The contents of the first 16 bits of the RTP data header extension structure must be defined if use of that mechanism is to be allowed under the profile for implementation-specific extensions (Section 5.3.1, p. 14).

**RTCP packet types:** New application-class-specific RTCP packet types may be defined and registered with IANA.

**RTCP report interval:** A profile should specify that the values suggested in Section 6.2 for the constants employed in the calculation of the RTCP report interval will be used. Those are the RTCP fraction of session bandwidth, the minimum report interval, and the bandwidth split between senders and receivers. A profile may specify alternate values if they have been demonstrated to work in a scalable manner.

**SR/RR extension:** An extension section may be defined for the RTCP SR and RR packets if there is additional information that should be reported regularly about the sender or receivers (Section 6.4.3, p. 32).

**SDES use:** The profile may specify the relative priorities for RTCP SDES items to be transmitted or excluded entirely (Section 6.3.9); an alternate syntax or semantics for the CNAME item (Section 6.5.1); the format of the LOC item (Section 6.5.5); the semantics and use of the NOTE item (Section 6.5.7); or new SDES item types to be registered with IANA.

**Security:** A profile may specify which security services and algorithms should be offered by applications, and may provide guidance as to their appropriate use (Section 9, p. 47).

**String-to-key mapping:** A profile may specify how a user-provided password or pass phrase is mapped into an encryption key.

**Underlying protocol:** Use of a particular underlying network or transport layer protocol to carry RTP packets may be required.

**Transport mapping:** A mapping of RTP and RTCP to transport-level addresses, e.g., UDP ports, other than the standard mapping defined in Section 10, p. 49 may be specified.

**Encapsulation:** An encapsulation of RTP packets may be defined to allow multiple RTP data packets to be carried in one lower-layer packet or to provide framing over underlying protocols that do not already do so (Section 10, p. 49).

It is not expected that a new profile will be required for every application. Within one application class, it would be better to extend an existing profile rather than make a new one in order to facilitate interoperation among the applications since each will typically run under only one profile. Simple extensions such as the definition of additional payload type values or RTCP packet types may be accomplished by registering them through the Internet Assigned Numbers Authority and publishing their descriptions in an addendum to the profile or in a payload format specification.

# A Algorithms

We provide examples of C code for aspects of RTP sender and receiver algorithms. There may be other implementation methods that are faster in particular operating environments or have other advantages. These implementation notes are for informational purposes only and are meant to clarify the RTP specification.

The following definitions are used for all examples; for clarity and brevity, the structure definitions are only valid for 32-bit big-endian (most significant octet first) architectures. Bit fields are assumed to be packed tightly in big-endian bit order, with no additional padding. Modifications would be required to construct a portable implementation.

```
/*
 * rtp.h  --  RTP header file (RFC XXXX)
 */
#include <sys/types.h>

/*
 * The type definitions below are valid for 32-bit architectures and
 * may have to be adjusted for 16- or 64-bit architectures.
 */
typedef unsigned char  u_int8;
typedef unsigned short u_int16;
typedef unsigned int   u_int32;
typedef          short int16;
```

```
/*
 * Current protocol version.
 */
#define RTP_VERSION    2

#define RTP_SEQ_MOD (1<<16)
#define RTP_MAX_SDES 255      /* maximum text length for SDES */

typedef enum {
    RTCP_SR   = 200,
    RTCP_RR   = 201,
    RTCP_SDES = 202,
    RTCP_BYE  = 203,
    RTCP_APP  = 204
} rtcp_type_t;

typedef enum {
    RTCP_SDES_END   = 0,
    RTCP_SDES_CNAME = 1,
    RTCP_SDES_NAME  = 2,
    RTCP_SDES_EMAIL = 3,
    RTCP_SDES_PHONE = 4,
    RTCP_SDES_LOC   = 5,
    RTCP_SDES_TOOL  = 6,
    RTCP_SDES_NOTE  = 7,
    RTCP_SDES_PRIV  = 8
} rtcp_sdes_type_t;

/*
 * RTP data header
 */
typedef struct {
    unsigned int version:2;   /* protocol version */
    unsigned int p:1;         /* padding flag */
    unsigned int x:1;         /* header extension flag */
    unsigned int cc:4;        /* CSRC count */
    unsigned int m:1;         /* marker bit */
    unsigned int pt:7;        /* payload type */
    unsigned int seq:16;      /* sequence number */
    u_int32 ts;               /* timestamp */
    u_int32 ssrc;             /* synchronization source */
    u_int32 csrc[1];          /* optional CSRC list */
} rtp_hdr_t;
```

```
/*
 * RTCP common header word
 */
typedef struct {
    unsigned int version:2;   /* protocol version */
    unsigned int p:1;         /* padding flag */
    unsigned int count:5;     /* varies by packet type */
    unsigned int pt:8;        /* RTCP packet type */
    u_int16 length;           /* pkt len in words, w/o this word */
} rtcp_common_t;

/*
 * Big-endian mask for version, padding bit and packet type pair
 */
#define RTCP_VALID_MASK (0xc000 | 0x2000 | 0xfe)
#define RTCP_VALID_VALUE ((RTP_VERSION << 14) | RTCP_SR)

/*
 * Reception report block
 */
typedef struct {
    u_int32 ssrc;             /* data source being reported */
    unsigned int fraction:8;  /* fraction lost since last SR/RR */
    int lost:24;              /* cumul. no. pkts lost (signed!) */
    u_int32 last_seq;         /* extended last seq. no. received */
    u_int32 jitter;           /* interarrival jitter */
    u_int32 lsr;              /* last SR packet from this source */
    u_int32 dlsr;             /* delay since last SR packet */
} rtcp_rr_t;

/*
 * SDES item
 */
typedef struct {
    u_int8 type;              /* type of item (rtcp_sdes_type_t) */
    u_int8 length;            /* length of item (in octets) */
    char data[1];             /* text, not null-terminated */
} rtcp_sdes_item_t;

/*
 * One RTCP packet
 */
typedef struct {
    rtcp_common_t common;     /* common header */
    union {
```

```
        /* sender report (SR) */
        struct {
            u_int32 ssrc;     /* sender generating this report */
            u_int32 ntp_sec;  /* NTP timestamp */
            u_int32 ntp_frac;
            u_int32 rtp_ts;   /* RTP timestamp */
            u_int32 psent;    /* packets sent */
            u_int32 osent;    /* octets sent */
            rtcp_rr_t rr[1];  /* variable-length list */
        } sr;

        /* reception report (RR) */
        struct {
            u_int32 ssrc;     /* receiver generating this report */
            rtcp_rr_t rr[1];  /* variable-length list */
        } rr;

        /* source description (SDES) */
        struct rtcp_sdes {
            u_int32 src;      /* first SSRC/CSRC */
            rtcp_sdes_item_t item[1]; /* list of SDES items */
        } sdes;

        /* BYE */
        struct {
            u_int32 src[1];   /* list of sources */
            /* can't express trailing text for reason */
        } bye;
    } r;
} rtcp_t;

typedef struct rtcp_sdes rtcp_sdes_t;

/*
 * Per-source state information
 */
typedef struct {
    u_int16 max_seq;        /* highest seq. number seen */
    u_int32 cycles;         /* shifted count of seq. number cycles */
    u_int32 base_seq;       /* base seq number */
    u_int32 bad_seq;        /* last 'bad' seq number + 1 */
    u_int32 probation;      /* sequ. packets till source is valid */
    u_int32 received;       /* packets received */
    u_int32 expected_prior; /* packet expected at last interval */
    u_int32 received_prior; /* packet received at last interval */
```

```
    u_int32 transit;        /* relative trans time for prev pkt */
    u_int32 jitter;         /* estimated jitter */
    /* ... */
} source;
```

## A.1 RTP Data Header Validity Checks

An RTP receiver should check the validity of the RTP header on incoming packets since they might be encrypted or might be from a different application that happens to be misaddressed. Similarly, if encryption according to the method described in Section 9 is enabled, the header validity check is needed to verify that incoming packets have been correctly decrypted, although a failure of the header validity check (e.g., unknown payload type) may not necessarily indicate decryption failure.

Only weak validity checks are possible on an RTP data packet from a source that has not been heard before:

- RTP version field must equal 2.
- The payload type must be known, in particular it must not be equal to SR or RR.
- If the P bit is set, then the last octet of the packet must contain a valid octet count, in particular, less than the total packet length minus the header size.
- The X bit must be zero if the profile does not specify that the header extension mechanism may be used. Otherwise, the extension length field must be less than the total packet size minus the fixed header length and padding.
- The length of the packet must be consistent with CC and payload type (if payloads have a known length).

The last three checks are somewhat complex and not always possible, leaving only the first two which total just a few bits. If the SSRC identifier in the packet is one that has been received before, then the packet is probably valid and checking if the sequence number is in the expected range provides further validation. If the SSRC identifier has not been seen before, then data packets carrying that identifier may be considered invalid until a small number of them arrive with consecutive sequence numbers.

The routine `update_seq` shown below ensures that a source is declared valid only after `MIN_SEQUENTIAL` packets have been received in sequence. It also validates the sequence number `seq` of a newly received packet and updates the sequence state for the packet's source in the structure to which `s` points.

When a new source is heard for the first time, that is, its SSRC identifier is not in the table (see Section 8.2), and the per-source state is allocated for it, `s->probation` should be set to the number of sequential packets required before declaring a source valid (parameter `MIN_SEQUENTIAL`) and `s->max_seq` initialized to `seq-1`. A non-zero `s->probation` marks the source as not yet valid so the state may be discarded after a short timeout rather than a long one, as discussed in Section 6.2.1.

After a source is considered valid, the sequence number is considered valid if it is no more than `MAX_DROPOUT` ahead of `s->max_seq` nor more than `MAX_MISORDER` behind. If the new sequence number is ahead of `max_seq` modulo the RTP sequence number range (16 bits), but is smaller than `max_seq`, it has wrapped around and the (shifted) count of sequence number cycles is incremented. A value of one is returned to indicate a valid sequence number.

Otherwise, the value zero is returned to indicate that the validation failed, and the bad sequence number is stored. If the next packet received carries the next higher sequence number, it is considered the valid start of a new packet sequence presumably caused by an extended dropout or a source restart. Since multiple complete sequence number cycles may have been missed, the packet loss statistics are reset.

Typical values for the parameters are shown, based on a maximum misordering time of 2 seconds at 50 packets/second and a maximum dropout of 1 minute. The dropout parameter `MAX_DROPOUT` should be a small fraction of the 16-bit sequence number space to give a reasonable probability that new sequence numbers after a restart will not fall in the acceptable range for sequence numbers from before the restart.

```
void init_seq(source *s, u_int16 seq)
{
    s->base_seq = seq - 1;
    s->max_seq = seq;
    s->bad_seq = RTP_SEQ_MOD + 1;
    s->cycles = 0;
    s->received = 0;
    s->received_prior = 0;
    s->expected_prior = 0;
    /* other initialization */
}

int update_seq(source *s, u_int16 seq)
{
    u_int16 udelta = seq - s->max_seq;
    const int MAX_DROPOUT = 3000;
    const int MAX_MISORDER = 100;
    const int MIN_SEQUENTIAL = 2;

    /*
     * Source is not valid until MIN_SEQUENTIAL packets with
     * sequential sequence numbers have been received.
     */
    if (s->probation) {
        /* packet is in sequence */
        if (seq == s->max_seq + 1) {
            s->probation--;
            s->max_seq = seq;
            if (s->probation == 0) {
                init_seq(s, seq);
                s->received++;
                return 1;
            }
        } else {
            s->probation = MIN_SEQUENTIAL - 1;
            s->max_seq = seq;
```

```
        }
        return 0;
    } else if (udelta < MAX_DROPOUT) {
        /* in order, with permissible gap */
        if (seq < s->max_seq) {
            /*
             * Sequence number wrapped - count another 64K cycle.
             */
            s->cycles += RTP_SEQ_MOD;
        }
        s->max_seq = seq;
    } else if (udelta <= RTP_SEQ_MOD - MAX_MISORDER) {
        /* the sequence number made a very large jump */
        if (seq == s->bad_seq) {
            /*
             * Two sequential packets -- assume that the other side
             * restarted without telling us so just re-sync
             * (i.e., pretend this was the first packet).
             */
            init_seq(s, seq);
        }
        else {
            s->bad_seq = (seq + 1) & (RTP_SEQ_MOD-1);
            return 0;
        }
    } else {
        /* duplicate or reordered packet */
    }
    s->received++;
    return 1;
}
```

The validity check can be made stronger requiring more than two packets in sequence. The disadvantages are that a larger number of initial packets will be discarded and that high packet loss rates could prevent validation. However, because the RTCP header validation is relatively strong, if an RTCP packet is received from a source before the data packets, the count could be adjusted so that only two packets are required in sequence. If initial data loss for a few seconds can be tolerated, an application could choose to discard all data packets from a source until a valid RTCP packet has been received from that source.

Depending on the application and encoding, algorithms may exploit additional knowledge about the payload format for further validation. For payload types where the timestamp increment is the same for all packets, the timestamp values can be predicted from the previous packet received from the same source using the sequence number difference (assuming no change in payload type).

A strong "fast-path" check is possible since with high probability the first four octets in the header of a newly received RTP data packet will be just the same as that of the previous packet from the same SSRC except that the sequence number will have increased by one. Similarly, a single-entry cache may be used for faster SSRC lookups in applications where data is typically received from one source at a time.

## A.2 RTCP Header Validity Checks

The following checks can be applied to RTCP packets.

- RTP version field must equal 2.
- The payload type field of the first RTCP packet in a compound packet must be equal to SR or RR.
- The padding bit (P) should be zero for the first packet of a compound RTCP packet because padding should only be applied, if it is needed, to the last packet.
- The length fields of the individual RTCP packets must total to the overall length of the compound RTCP packet as received. This is a fairly strong check.

The code fragment below performs all of these checks. The packet type is not checked for subsequent packets since unknown packet types may be present and should be ignored.

```
u_int32 len;        /* length of compound RTCP packet in words */
rtcp_t *r;          /* RTCP header */
rtcp_t *end;        /* end of compound RTCP packet */

if ((*(u_int16 *)r & RTCP_VALID_MASK) != RTCP_VALID_VALUE) {
    /* something wrong with packet format */
}
end = (rtcp_t *)((u_int32 *)r + len);

do r = (rtcp_t *)((u_int32 *)r + r->common.length + 1);
while (r < end && r->common.version == 2);

if (r != end) {
    /* something wrong with packet format */
}
```

## A.3 Determining the Number of RTP Packets Expected and Lost

In order to compute packet loss rates, the number of packets expected and actually received from each source needs to be known, using per-source state information defined in `struct source` referenced via pointer `s` in the code below. The number of packets received is simply the count of packets as they arrive, including any late or duplicate packets. The number of packets expected can be computed by the receiver as the difference between the highest sequence number received (`s->max_seq`) and the first sequence number received (`s->base_seq`). Since the sequence number is only 16 bits and will wrap around, it is necessary to extend the highest sequence number with the (shifted) count of sequence number wraparounds (`s->cycles`). Both the received packet count and the count of cycles are maintained the RTP header validity check routine in Appendix A.1.

```
extended_max = s->cycles + s->max_seq;
expected = extended_max - s->base_seq + 1;
```

The number of packets lost is defined to be the number of packets expected less the number of packets actually received:

```
lost = expected - s->received;
```

Since this signed number is carried in 24 bits, it SHOULD be clamped at 0x7fffff for positive loss or 0xffffff for negative loss rather than wrapping around.

The fraction of packets lost during the last reporting interval (since the previous SR or RR packet was sent) is calculated from differences in the expected and received packet counts across the interval, where `expected_prior` and `received_prior` are the values saved when the previous reception report was generated:

```
expected_interval = expected - s->expected_prior;
s->expected_prior = expected;
received_interval = s->received - s->received_prior;
s->received_prior = s->received;
lost_interval = expected_interval - received_interval;
if (expected_interval == 0 || lost_interval <= 0) fraction = 0;
else fraction = (lost_interval << 8) / expected_interval;
```

The resulting `fraction` is an 8-bit fixed point number with the binary point at the left edge.

## A.4 Generating SDES RTCP Packets

This function builds one SDES chunk into buffer `b` composed of `argc` items supplied in arrays `type`, `value` and `length`. It returns a pointer to next available location within `b`.

```
char *rtp_write_sdes(char *b, u_int32 src, int argc,
                     rtcp_sdes_type_t type[], char *value[],
     int length[])
{
    rtcp_sdes_t *s = (rtcp_sdes_t *)b;
    rtcp_sdes_item_t *rsp;
    int i;
    int len;
    int pad;

    /* SSRC header */
    s->src = src;
    rsp = &s->item[0];

    /* SDES items */
    for (i = 0; i < argc; i++) {
        rsp->type = type[i];
        len = length[i];
```

```
        if (len > RTP_MAX_SDES) {
            /* invalid length, may want to take other action */
            len = RTP_MAX_SDES;
        }
        rsp->length = len;
        memcpy(rsp->data, value[i], len);
        rsp = (rtcp_sdes_item_t *)&rsp->data[len];
    }

    /* terminate with end marker and pad to next 4-octet boundary */
    len = ((char *) rsp) - b;
    pad = 4 - (len & 0x3);
    b = (char *) rsp;
    while (pad--) *b++ = RTCP_SDES_END;

    return b;
}
```

## A.5 Parsing RTCP SDES Packets

This function parses an SDES packet, calling functions `find_member()` to find a pointer to the information for a session member given the SSRC identifier and `member_sdes()` to store the new SDES information for that member. This function expects a pointer to the header of the RTCP packet.

```
void rtp_read_sdes(rtcp_t *r)
{
    int count = r->common.count;
    rtcp_sdes_t *sd = &r->r.sdes;
    rtcp_sdes_item_t *rsp, *rspn;
    rtcp_sdes_item_t *end = (rtcp_sdes_item_t *)
                            ((u_int32 *)r + r->common.length + 1);
    source *s;

    while (--count >= 0) {
        rsp = &sd->item[0];
        if (rsp >= end) break;
        s = find_member(sd->src);

        for (; rsp->type; rsp = rspn ) {
            rspn = (rtcp_sdes_item_t *)((char*)rsp+rsp->length+2);
            if (rspn >= end) {
                rsp = rspn;
                break;
            }
            member_sdes(s, rsp->type, rsp->data, rsp->length);
        }
```

```
        sd = (rtcp_sdes_t *)
             ((u_int32 *)sd + (((char *)rsp - (char *)sd) >> 2)+1);
    }
    if (count >= 0) {
        /* invalid packet format */
    }
}
```

## A.6 Generating a Random 32-bit Identifier

The following subroutine generates a random 32-bit identifier using the MD5 routines published in RFC 1321 [24]. The system routines may not be present on all operating systems, but they should serve as hints as to what kinds of information may be used. Other system calls that may be appropriate include

- `getdomainname()`,
- `getwd()`, or
- `getrusage()`.

"Live" video or audio samples are also a good source of random numbers, but care must be taken to avoid using a turned-off microphone or blinded camera as a source [8].

Use of this or similar routine is suggested to generate the initial seed for the random number generator producing the RTCP period (as shown in Appendix A.7), to generate the initial values for the sequence number and timestamp, and to generate SSRC values. Since this routine is likely to be CPU-intensive, its direct use to generate RTCP periods is inappropriate because predictability is not an issue. Note that this routine produces the same result on repeated calls until the value of the system clock changes unless different values are supplied for the `type` argument.

```
/*
 * Generate a random 32-bit quantity.
 */
#include <sys/types.h>   /* u_long */
#include <sys/time.h>    /* gettimeofday() */
#include <unistd.h>      /* get..() */
#include <stdio.h>       /* printf() */
#include <time.h>        /* clock() */
#include <sys/utsname.h> /* uname() */
#include "global.h"      /* from RFC 1321 */
#include "md5.h"         /* from RFC 1321 */

#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final

static u_long md_32(char *string, int length)
```

```
{
    MD_CTX context;
    union {
        char   c[16];
        u_long x[4];
    } digest;
    u_long r;
    int i;

    MDInit (&context);
    MDUpdate (&context, string, length);
    MDFinal ((unsigned char *)&digest, &context);
    r = 0;
    for (i = 0; i < 3; i++) {
        r ^= digest.x[i];
    }
    return r;
}                               /* md_32 */


/*
 * Return random unsigned 32-bit quantity. Use 'type' argument if you
 * need to generate several different values in close succession.
 */
u_int32 random32(int type)
{
    struct {
        int     type;
        struct  timeval tv;
        clock_t cpu;
        pid_t   pid;
        u_long  hid;
        uid_t   uid;
        gid_t   gid;
        struct  utsname name;
    } s;

    gettimeofday(&s.tv, 0);
    uname(&s.name);
    s.type = type;
    s.cpu  = clock();
    s.pid  = getpid();
    s.hid  = gethostid();
    s.uid  = getuid();
    s.gid  = getgid();
```

```
    /* also: system uptime */

    return md_32((char *)&s, sizeof(s));
}                                /* random32 */
```

### A.7 Computing the RTCP Transmission Interval

The following functions implement the RTCP transmission and reception rules described in Section 6.2. These rules are coded in several functions:

- `OnExpire()` is called when the RTCP transmission timer expires.
- `rtcp_interval()` computes the deterministic calculated interval, measured in seconds.
- `OnReception()` is called whenever an RTCP packet is received.

It is assumed that the following functions are available:

- Schedule(time t, event e) schedules an event e to occur at time t. When time t arrives, the funcion OnExpire is called with e as an argument.
- ReSchedule(time t, event e) reschedules a previously scheduled event e for time t.
- SendRTCPReport() sends an RTCP report.
- SendBYEPacket() sends a BYE packet.
- TypeOfEvent(event e) returns EVENT_BYE if the event being processed is for a BYE packet to be sent, else it returns EVENT_REPORT.
- NewMember(p) returns a 1 if the person who sent packet p is not currently in the member list, 0 otherwise. Note this function is not sufficient for a complete implementation because each CSRC identifier in an RTP packet and each SSRC in a BYE packet should be processed.
- PacketType(p) returns PACKET_RTCP_REPORT if packet p is an RTCP report (not BYE), PACKET_BYE if its a BYE RTCP packet, and PACKET_RTP if its a regular RTP data packet.

The parameters of `rtcp_interval()` are defined in Section 6.3.

```
double rtcp_interval(int members,
                     int senders,
                     double rtcp_bw,
                     int we_sent,
                     double avg_rtcp_size,
                     int initial)
{
    /*
     * Minimum average time between RTCP packets from this site (in
     * seconds).  This time prevents the reports from `clumping' when
```

```
     * sessions are small and the law of large numbers isn't helping
     * to smooth out the traffic.  It also keeps the report interval
     * from becoming ridiculously small during transient outages like
     * a network partition.
     */
    double const RTCP_MIN_TIME = 5.;
    /*
     * Fraction of the RTCP bandwidth to be shared among active
     * senders.  (This fraction was chosen so that in a typical
     * session with one or two active senders, the computed report
     * time would be roughly equal to the minimum report time so that
     * we don't unnecessarily slow down receiver reports.) The
     * receiver fraction must be 1 - the sender fraction.
     */
    double const RTCP_SENDER_BW_FRACTION = 0.25;
    double const RTCP_RCVR_BW_FRACTION = (1-RTCP_SENDER_BW_FRACTION);
    double t;                   /* interval */
    double rtcp_min_time = RTCP_MIN_TIME;
    int n;                      /* no. of members for computation */

    /*
     * Very first call at application start-up uses half the min
     * delay for quicker notification while still allowing some time
     * before reporting for randomization and to learn about other
     * sources so the report interval will converge to the correct
     * interval more quickly.
     */
    if (initial) {
        rtcp_min_time /= 2;
    }

    /*
     * If there were active senders, give them at least a minimum
     * share of the RTCP bandwidth.  Otherwise all participants share
     * the RTCP bandwidth equally.
     */
    n = members;
    if (senders > 0 && senders < members * RTCP_SENDER_BW_FRACTION) {
        if (we_sent) {
            rtcp_bw *= RTCP_SENDER_BW_FRACTION;
            n = senders;
        } else {
            rtcp_bw *= RTCP_RCVR_BW_FRACTION;
            n -= senders;
        }
```

```
    }

    /*
     * The effective number of sites times the average packet size is
     * the total number of octets sent when each site sends a report.
     * Dividing this by the effective bandwidth gives the time
     * interval over which those packets must be sent in order to
     * meet the bandwidth target, with a minimum enforced.  In that
     * time interval we send one report so this time is also our
     * average time between reports.
     */
    t = avg_rtcp_size * n / rtcp_bw;
    if (t < rtcp_min_time) t = rtcp_min_time;

    /*
     * To avoid traffic bursts from unintended synchronization with
     * other sites, we then pick our actual next report interval as a
     * random number uniformly distributed between 0.5*t and 1.5*t.
     */
    return t * (drand48() + 0.5);
}

void OnExpire(event e,
              int    members,
              int    senders,
              double rtcp_bw,
              int    we_sent,
              double *avg_rtcp_size,
              int    *initial,
              time   tc,
              time   *tp,
              int    *pmembers) {

    /* This function is responsible for deciding whether to send
     * an RTCP report or BYE packet now, or to reschedule transmission.
     * It is also responsible for updating the pmembers, initial, tp,
     * and avg_rtcp_size state variables. This function should be called
     * upon expiration of the event timer used by Schedule(). */

    double t;     /* Interval */
    double tn;    /* Next transmit time */
    int SendIt;   /* flag for sending packet */

    /* To compensate for OPTION B converging to a value below the
     * intended average. */
```

```
    double const COMPENSATION = 2.71828 - 1.5;

    /* In the case of a BYE, we use OPTION B to reschedule the
     * transmission of the BYE if necessary */

    if(TypeOfEvent(e) == EVENT_BYE) {
        t = rtcp_interval(members,
                          senders,
                          rtcp_bw * COMPENSATION,
                          we_sent,
                          avg_rtcp_size,
                          initial);
        tn = *tp + t;
        if(tn <= tc) {
           SendBYEPacket();
           exit(1);
        } else {
           Schedule(tn, e);
        }

    } else if(TypeOfEvent(e) == EVENT_REPORT) {
        SendIt = FALSE;
        if((algorithm == ALGORITHM_A) ||
           ((algorithm == ALGORITHM_C) && (initial == FALSE))) {
           t = rtcp_interval(members,
                             senders,
                             rtcp_bw,
                             we_sent,
                             avg_rtcp_size,
                             initial);

           if(members <= pmembers) {
              SendIt = TRUE;
           } else {
              tn = *tp + t;

              if(tn <= tc) {
                 SendIt = TRUE;
              }
           }
        } else if((algorithm == ALGORITHM_B) ||
                  ((algorithm == ALGORITHM_C) && (initial == TRUE))) {
           t = rtcp_interval(members,
                             senders,
                             rtcp_bw * COMPENSATION,
```

```
                             we_sent,
                             avg_rtcp_size,
                             initial);

            tn = *tp + t;

            if(tn <= tc) {
               SendIt = TRUE;
            }
         }

         if(SendIt == TRUE) {
             SendRTCPReport();
             *pmembers = members;
             *avg_rtcp_size = (1./16.)*PacketSize(e) +
                 (15./16.)*(*avg_rtcp_size);
             *tp = tc;
         } else {
             Schedule(tn, e);
             *pmembers = members;
         }
     }
}


void OnReceive(packet p,
               event e,
               int *members,
               int *pmembers,
               int *senders
               double *avg_rtcp_size,
               double *tp,
               double tc) {

  double tn;  /* Next packet transmission time */

  /* What we do depends on whether we have left the group, and
   * are waiting to send a BYE (TypeOfEvent(e) == EVENT_BYE) or
   * an RTCP report. p represents the packet that was just received. */

  if(PacketType(p) == PACKET_RTCP_REPORT) {
    if(NewMember(p) && (TypeOfEvent(e) == EVENT_REPORT)) *members += 1;
    *avg_rtcp_size = (1./16.)*PacketSize(e) + (15./16.)*(*avg_rtcp_size);
  } else if(PacketType(p) == PACKET_RTP) {
    if(NewSender(p) && (TypeOfEvent(e) == EVENT_REPORT)) *senders += 1;
```

```
    if(NewMember(p) && (TypeOfEvent(e) == EVENT_REPORT)) *members += 1;
  } else if(PacketType(p) == PACKET_BYE) {
    *avg_rtcp_size = (1./16.)*PacketSize(e) + (15./16.)*(*avg_rtcp_size);

    if(TypeOfEvent(e) == EVENT_REPORT) {
      if(NewSender(p) == FALSE) *senders -= 1;
      if(NewMember(p) == FALSE) *members -= 1;

      tn = tc + ((*members)/(*pmembers))*(tn - tc);
      *tp = tc - ((*members)/(*pmembers))*(tc - *tp);

      /* Reschedule the next report for time tn */

      Reschedule(e, tn);
      *pmembers = members;

    } else if(TypeOfEvent(e) == EVENT_BYE) {

      *members += 1;

    }
  }
}
```

### A.8 Estimating the Interarrival Jitter

The code fragments below implement the algorithm given in Section 6.4.1 for calculating an estimate of the statistical variance of the RTP data interarrival time to be inserted in the interarrival jitter field of reception reports. The inputs are `r->ts`, the timestamp from the incoming packet, and `arrival`, the current time in the same units. Here `s` points to state for the source; `s->transit` holds the relative transit time for the previous packet, and `s->jitter` holds the estimated jitter. The jitter field of the reception report is measured in timestamp units and expressed as an unsigned integer, but the jitter estimate is kept in a floating point. As each data packet arrives, the jitter estimate is updated:

```
int transit = arrival - r->ts;
int d = transit - s->transit;
s->transit = transit;
if (d < 0) d = -d;
s->jitter += (1./16.) * ((double)d - s->jitter);
```

When a reception report block (to which `rr` points) is generated for this member, the current jitter estimate is returned:

```
rr->jitter = (u_int32) s->jitter;
```

Alternatively, the jitter estimate can be kept as an integer, but scaled to reduce round-off error. The calculation is the same except for the last line:

```
s->jitter += d - ((s->jitter + 8) >> 4);
```

In this case, the estimate is sampled for the reception report as:

```
rr->jitter = s->jitter >> 4;
```

# B Changes from RFC 1889

Most of this RFC is identical to RFC 1889. The changes are listed below.

- The algorithm for calculating the RTCP transmission interval specified in Sections 6.2 and 6.3 and illustrated in Appendix A.7 is augmented to include "reconsideration" to minimize transmission over the intended rate when many participants join a session simultaneously, and "reverse reconsideration" to reduce the incidence and duration of false participant timeouts when the number of participants drops rapidly.
- Section 6.3.7 specifies new rules controlling when an RTCP BYE packet should be sent in order to avoid a flood of packets when many participants leave a session simultaneously. Sections 7.2 and 7.3 specify that translators and mixers should send BYE packets for the sources they are no longer forwarding.
- An algorithm is specified in Section 6.2.1 to allow storage of only a sampling of the participants' SSRC identifiers to allow scaling to very large sessions.
- In Section 6.2 it is specified that RTCP sender and receiver bandwidths to be set as separate parameters of the session rather than a strict percentage of the session bandwidth.
- Also in Section 6.2 it is specified that the minimum RTCP interval may be scaled to smaller values for high bandwidth sessions.
- Rule changes for layered encodings are defined in Sections 2.4, 6.3.9, 8.3 and 10.
- An indentation bug in the RFC 1889 printing of the pseudo-code for the collision detection and resolution algorithm in Section 8.2 is corrected, and the algorithm has been modified to remove the restriction that both RTP and RTCP must be sent from the same source port number.
- For unicast RTP sessions, distinct port pairs may be used for the two ends (Sections 3 and 7.1).
- The descriptin of the padding mechanism for RTCP packets was clarified and it is specified that padding MUST be applied to the last packet of a compound RTCP packet.
- It is specified that a receiver MUST ignore packets with payload types it does not understand.

- The specification of "relative" NTP timestamp in the RTCP SR section now defines these timestamps to be based on the most common system-specific, clock such as system uptime, rather than on session elapsed time which would not be the same for multiple applications started on the same machine but at different times.
- The reference for the UTF-8 character set was changed to be RFC 2279.
- Small clarifications of the text have been made in several places, some in response to questions from readers. In particular:
  - A definition for "RTP media type" is given in Section 3 to allow the explanation of multiplexing RTP sessions in Section 5.2 to be more clear regarding the multiplexing of multiple media.
  - The description of the session bandwidth parameter is expanded in Section 6.2.
  - The method for padding RTCP packets is clarified in Section 6.4.
  - The method for terminating and padding a sequence of SDES items is clarified in Section 6.5.
  - The Security section adds a formal reference to IPSEC now that it is available, and says that the confidentiality method defined in this specification is primarily to codify existing practice.
  - The terms MUST, SHOULD, MAY, etc. are used as defined in RFC 2119.

# C Security Considerations

RTP suffers from the same security liabilities as the underlying protocols. For example, an impostor can fake source or destination network addresses, or change the header or payload. Within RTCP, the CNAME and NAME information may be used to impersonate another participant. In addition, RTP may be sent via IP multicast, which provides no direct means for a sender to know all the receivers of the data sent and therefore no measure of privacy. Rightly or not, users may be more sensitive to privacy concerns with audio and video communication than they have been with more traditional forms of network communication [25]. Therefore, the use of security mechanisms with RTP is important. These mechanisms are discussed in Section 9.

RTP-level translators or mixers may be used to allow RTP traffic to reach hosts behind firewalls. Appropriate firewall security principles and practices, which are beyond the scope of this document, should be followed in the design and installation of these devices and in the admission of RTP applications for use behind the firewall.

# D Addresses of Authors

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: `schulzrinne@cs.columbia.edu`

Stephen L. Casner
Cisco Systems, Inc.
1072 Arastradero Road
Palo Alto, CA 94304
United States
electronic mail: `casner@cisco.com`

Ron Frederick
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
United States
electronic mail: `frederic@parc.xerox.com`

Van Jacobson
MS 46a-1121
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
United States
electronic mail: `van@ee.lbl.gov`

## Acknowledgments

This memorandum is based on discussions within the IETF Audio/Video Transport working group chaired by Stephen Casner. The current protocol has its origins in the Network Voice Protocol and the Packet Video Protocol (Danny Cohen and Randy Cole) and the protocol implemented by the vat application (Van Jacobson and Steve McCanne). Christian Huitema provided ideas for the random identifier generator. Extensive analysis and simulation of the timer reconsideration algorithm was done by Jonathan Rosenberg.

## References

[1] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Philadelphia, Pennsylvania), pp. 200–208, IEEE, Sept. 1990. Computer Communications Review, Vol. 20(4), Sept. 1990.

[2] H. Schulzrinne, "Issues in designing a transport protocol for audio and video conferences and other multiparticipant real-time applications." expired Internet draft, Oct. 1993.

[3] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, Internet Engineering Task Force, Mar. 1997.

[4] D. E. Comer, *Internetworking with TCP/IP*, vol. 1. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[5] J. Postel, "Internet protocol," RFC 791, Internet Engineering Task Force, Sept. 1981.

[6] D. Mills, "Network time protocol (v3)," RFC 1305, Internet Engineering Task Force, Apr. 1992.

[7] J. Reynolds and J. Postel, "Assigned numbers," STD 2, RFC 1700, Internet Engineering Task Force, Oct. 1994.

[8] D. Eastlake, S. Crocker, and J. Schiller, "Randomness recommendations for security," RFC 1750, Internet Engineering Task Force, Dec. 1994.

[9] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, England), pp. 58–67, ACM, Aug. 1994.

[10] I. Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP," *Computer Communications*, Jan. 1996.

[11] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (San Francisco, California), pp. 33–44, ACM, Sept. 1993. also in [26].

[12] J. A. Cadzow, *Foundations of digital signal processing and data analysis*. New York, New York: Macmillan, 1987.

[13] F. Yergeau, "UTF-8, a transformation format of ISO 10646," RFC 2279, Internet Engineering Task Force, Jan. 1998.

[14] P. Mockapetris, "Domain names - concepts and facilities," STD 13, RFC 1034, Internet Engineering Task Force, Nov. 1987.

[15] P. Mockapetris, "Domain names - implementation and specification," STD 13, RFC 1035, Internet Engineering Task Force, Nov. 1987.

[16] R. Braden, "Requirements for internet hosts - application and support," STD 3, RFC 1123, Internet Engineering Task Force, Oct. 1989.

[17] Y. Rekhter, R. Moskowitz, D. Karrenberg, and G. de Groot, "Address allocation for private internets," RFC 1597, Internet Engineering Task Force, Mar. 1994.

[18] E. Lear, E. Fair, D. Crocker, and T. Kessler, "Network 10 considered harmful (some practices shouldn't be codified)," RFC 1627, Internet Engineering Task Force, July 1994.

[19] D. Crocker, "Standard for the format of ARPA internet text messages," STD 11, RFC 822, Internet Engineering Task Force, Aug. 1982.

[20] W. Feller, *An Introduction to Probability Theory and its Applications, Volume 1*, vol. 1. New York, New York: John Wiley and Sons, third ed., 1968.

[21] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," Internet Draft, Internet Engineering Task Force, July 1998. Work in progress.

[22] D. Balenson, "Privacy enhancement for internet electronic mail: Part III: algorithms, modes, and identifiers," RFC 1423, Internet Engineering Task Force, Feb. 1993.

[23] V. L. Voydock and S. T. Kent, “Security mechanisms in high-level network protocols,” *ACM Computing Surveys*, vol. 15, pp. 135–171, June 1983.

[24] R. Rivest, “The MD5 message-digest algorithm,” RFC 1321, Internet Engineering Task Force, Apr. 1992.

[25] S. Stubblebine, “Security services for multimedia conferencing,” in *16th National Computer Security Conference*, (Baltimore, Maryland), pp. 391–395, Sept. 1993.

[26] S. Floyd and V. Jacobson, “The synchronization of periodic routing messages,” *IEEE/ACM Transactions on Networking*, vol. 2, pp. 122–136, Apr. 1994.