Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-avt-rtp-05.ps

Audio-Video Transport WG
Schulzrinne/Casner/Frederick/Jacobson
GMD/ISI/Xerox/LBL
July 18, 1994
Expires: 10/1/94

RTP: A Transport Protocol for Real-Time Applications

Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

Distribution of this document is unlimited.

Abstract

This memorandum describes the real-time transport protocol, RTP. 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) designed to provide minimal control and identification functionality, particularly in multicast networks. 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 bridges.

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.

The protocol is under development and changes to aspects of the protocol are likely. The presentation of protocol aspects does not imply that all members of the working group agree with the particular choice. Items to be discussed are marked with TBD.

Contents

1 Introduction

This memorandum specifies the real-time transport protocol (RTP), which provides end-to-end delivery services for data with real-time characteristics, for example, interactive audio and video. 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 end system 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. RTP is designed to run on top of a variety of network and transport protocols, for example, IP, ST-II or UDP.[1] RTP transfers data in a single direction, possibly to multiple destinations if supported by the underlying network.

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. Profiles are used to instantiate certain parts of the header for particular sets of applications (see Section 9). A profile for audio and video data may be found in the companion Internet draft `draft-ietf-avt-profile`[2].

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

- the real-time transport protocol (RTP), for exchanging data that has real-time properties.
- the RTP control protocol (RTCP), for monitoring quality of service and for conveying information about the participants in an on-going session. The latter aspect of RTCP is used for "loosely controlled" sessions, i.e., where there is no explicit membership control and set-up. This functionality may be fully or partially subsumed by a session control protocol, which is beyond the scope of this document.

A discussion of real-time services and algorithms for their implementation and background on some of the RTP design decisions can be found in the current version of the companion Internet draft `draft-ietf-avt-issues`.

The current Internet does not support the widespread use of real-time services. High-bandwidth services using RTP, such as video, can potentially seriously degrade other network services. Thus, implementors should take appropriate precautions to limit accidental bandwidth usage. Application documentation should clearly outline the limitations and possible operational impact of high-bandwidth real-time services on the Internet and other network services.

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 Internet draft `draft-ietf-avt-profile`.

[1] For most applications, RTP offers insufficient demultiplexing to run directly on IP.

[2] ftp://ds.internic.net/internet-draft/draft-ietf-avt-profile-03.txt

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 control (RTCP) packets, and the other is used for audio data. This address and port information is distributed to the intended participants. The exact details of the allocation and distribution mechanism 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 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 seen by the source, so that, in this example, a chunk of audio is delivered to the speaker every 20 ms. The sequence number can also be used by the receiver to estimate how many packets are being lost. Each RTP packet also indicates what type of audio encoding (such as PCM, ADPCM or GSM) is being used, 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.

Each audio source has to have its timing reconstructed separately at the receiver. Sources are identified by the synchronization source identifier (SSRC), not their network address. The SSRC identifier is a randomly chosen value meant to be globally unique within a particular conference.

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 email address and other user information may also be included. A site sends the RTCP BYE (Section 6.6) chunk when it leaves a conference. The RTCP reception report indicates how well the current speaker is being received and may be used to control adaptive encodings.

2.2 Bridges

So far, we have assumed that all sites want to receive audio 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, a *bridge* is placed near the low-bandwidth area. This bridge resynchronizes incoming audio packets to reconstruct the constant 20 ms spacing generated by the sender, mixes these reconstructed audio streams, translates the audio encoding to a lower-bandwidth one and forwards the lower-bandwidth packet stream to the low-bandwidth sites.

Since the bridge has constructed a new (mixed) stream of audio, it is now the synchronization source for the stream. In order to preserve the identity of the sites which are speaking, the bridge inserts one or more content source (CSRC) items after the fixed RTP header. These items contain the synchronization source identifiers (SSRC) of the site(s) that contributed to the mixed packet. An example of this is shown for bridge B1 in Fig. 1. As name and location information is received by the bridge in RTCP chunks from the high-speed sites, that information is passed on to the receivers served by the mixer, either aggregated or as received.

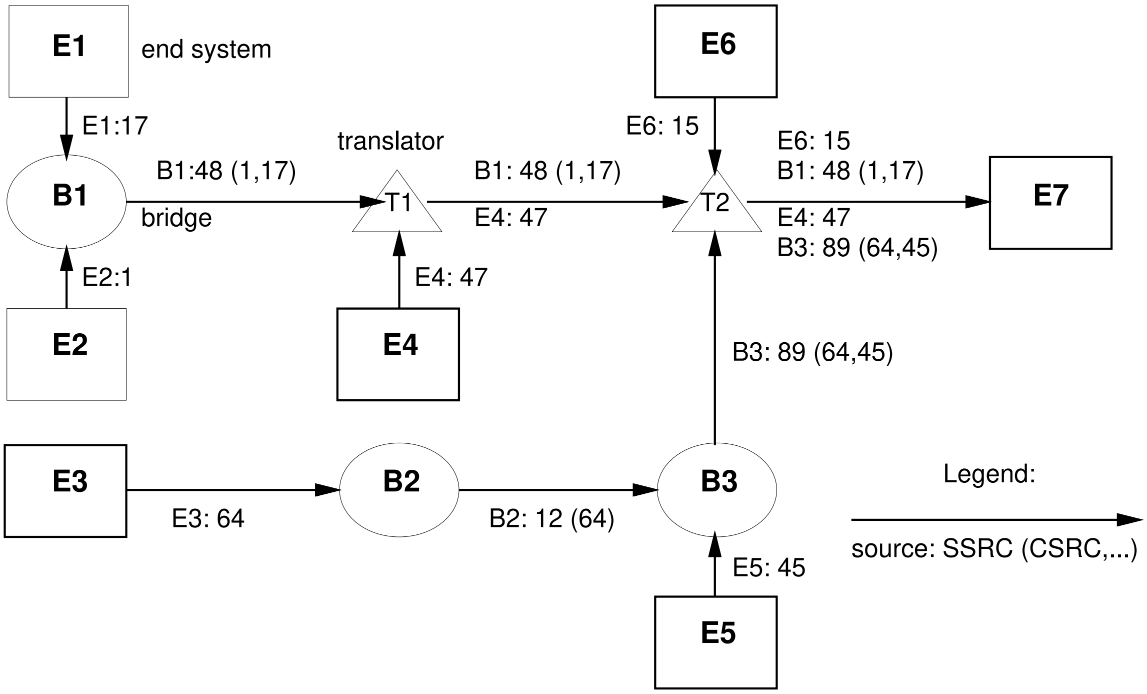

Figure 1: Sample RTP network with end systems, bridges and translators

2.3 Translators

Not all sites are directly accessible through IP multicast. For these sites, mixing may not be necessary, but a translation of the underlying transport protocol is. RTP-level gateways that do not restore timing or mix packets from different sources are called *translators* in this document. Application-level firewalls, for example, will not let any IP packets pass. Two translators are installed, one on either side of the firewall, the outside one funneling all multicast packets received through the 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. Other examples include the connection of a group of hosts speaking only IP/UDP to a group of hosts that understand only ST-II. The packet-by-packet encoding translation of single sources is another example.

The SSRC identifier makes it possible to identify individual sources even though they all pass through the same translator, i.e., carry the same network source address. In Fig. 1, hosts T1 and T2 are translators.

2.4 Security

Conference participants would often like to ensure that nobody else can listen to their deliberations. Encryption provides that privacy. In Section 7.1, RTP specifies a mechanism for using encryption, but the actual key distribution must be accomplished by external means.

3 Definitions

RTP payload is the data following the RTP fixed header and the CSRC list. The payload format and interpretation are beyond the scope of this memo. Examples of payload include audio samples and video data.

An *RTP packet* consists the fixed RTP header, a possibly empty list of content sources (CSRC list), and the payload, if any. Some underlying protocols may require an encapsulation of the RTP packet to be defined. A single packet of the underlying protocol may contain several RTP packets if permitted by the encapsulation method.

A *(protocol) port* is 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." [1] The transport selectors (TSEL) used by the OSI transport layer are equivalent to ports.

A *content source* is the actual source of the data carried in an RTP packet, for example, the application that originally generated some audio data. Data from one or more content sources may be combined into a single RTP packet by a bridge, which becomes the synchronization source (see next paragraph). Content source identifiers carried in the CSRC list identify the logical source of the data, for example, to highlight the current speaker in an audio conference; they have no effect on the delivery or playout timing of the data itself. In Fig. 1, E1 and E2 are the content sources of the data received by E7 from bridge B1, while B1 is the synchronization source.

A *synchronization source* may be a single content source, or the combination (mixing) of one or more content sources, produced by a bridge, with its own timing. Each synchronization source has its own sequence number space. The packetized audio coming from a single microphone, the mixed audio from an analog mixer or an RTP bridge and the video from a camera are examples of synchronization sources. The receiver groups packets by synchronization source for playback. Typically a single synchronization source emits a single medium (e.g., audio or video). A synchronization source may change its data format, e.g., audio encoding, over time. Synchronization sources are identified by the SSRC value in the RTP header.

An *end system* generates the content to be sent in RTP packets and consumes the content of received RTP packets. An end system can act as one or more synchronization sources. (Most end systems are expected to be a single synchronization source.) When a packet is transmitted from an end system, the end system is the content source, synchronization source, and transport source at that point.

An (RTP-level) *bridge* receives RTP packets from one or more sources, combines them in some manner and then forwards a new RTP packet. A bridge may change the data format. Since the timing among multiple input sources will not generally be synchronized, the bridge will make timing adjustments among the streams and generate its own timing for the combined stream. Therefore, when a packet is processed through a bridge, the bridge becomes the synchronization source as well as the transport source, but the originating end systems remain the content sources for that data. As the bridge combines packets from multiple content sources into a single outgoing packet, each of the contributing content sources is noted by the insertion of an identifier into the CSRC list in the outgoing packet. If a bridge receives data from another bridge, only the CSRC list should be copied to the outgoing packet; the SSRC of the bridge is not inserted in the outgoing CSRC list. Audio bridges and media converters are examples of bridges. In Fig. 1, end systems E1 and E2 use the services of bridge B1. B1 inserts CSRC identifiers for E1 and E2 when they are active (e.g., talking in an audio conference). The RTP-level bridges described in this document are unrelated to the data link-layer bridges found in local area networks. If there is possibility for confusion, the term 'RTP-level bridge' should be used. The name "bridge" follows common telecommunications industry usage.

An (RTP-level) *translator* forwards RTP packets, but does not alter their sequence numbers or timestamps. Examples of translators include devices that convert encodings without mixing or retiming or convert from multicast to unicast, and application-level filters in firewalls. A translator is a transport source, but neither a synchronization nor a content source.

A *(QOS) monitor* is an application that receives RTCP messages, including quality-of-service reports, and estimates the current quality of service for monitoring, fault diagnosis and long-term statistics.

A *recorder* records RTP and RTCP packets for later playback. It should try to recreate the timing at the sender, without the jitter introduced by the network, using the RTP timestamp. A recorder may not have access to the same encryption keys as the other participants in a session, in which case sender timing must be estimated if the RTP timestamps are encrypted.

Non-RTP mechanisms refers to other protocols and mechanisms that may be needed to provide a usable service. In particular, for multimedia conferences, a conference control application may distribute multicast addresses and keys for encryption and authentication, negotiate the encryption algorithm to be used, and determine the mapping from the RTP format field to the actual data format used. For simple applications, electronic mail or a conference database may also be used. The specification of such mechanisms is outside the scope of this memorandum.

4 Byte Order, Alignment, and Reserved Values

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 [2], 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 words are aligned on even byte addresses, 32-bit long words are aligned at addresses divisible by four, etc. Octets designated as padding have the value zero. Fields designated as "reserved" or `R` are set aside for future use; they should be set to zero by senders and ignored by receivers.

Textual information is encoded according to the UTF-2 encoding specified in Annex F of ISO standard 10646 [3,4]. 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 to a value of one. Strings may be padded with octets with a binary value of zero, but no padding or other string termination is required.

NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0h UTC on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits [5].

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source identifier (SSRC)            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|              content source identifiers (CSRCs)               |
|                             ....                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

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

type (T): 2 bits
Identifies the type of RTP packet. The type of the packet described here is two (2). (The value of 2 was chosen to easily distinguish packets from those of the prior version of RTP and the protocol used by the vat audio tool.)

padding (P): 1 bit
If the padding bit is one, the packet contains one or more additional octets at the end which are not part of the payload. The very last octet of the packet is a count of how many padding octets should be ignored. 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
The bit indicates that the fixed header is followed by exactly one header extension, with a format defined in Section 5.3.

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

marker (M): 1 bit
The interpretation of this field is defined by a profile. A profile may define additional marker bits by reducing the number of bits in the payload type field.

payload type (PT): 7 bits
The payload type forms an index into a table defined through profiles, the RTCP FMT packet (see Section 6.7) and non-RTP mechanisms (see Section 3). The mapping establishes the format of the RTP payload and determines its interpretation by the application. A profile specifies a standard mapping. An initial set of default mappings for audio and video is specified in the companion profile document RFC TBD, and may be extended in future editions of the Assigned Numbers RFC.

sequence number: 16 bits
The sequence number counts RTP packets. The sequence number increments by one for each packet sent. The sequence number may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number is random (unpredictable) to make known-plaintext attacks on encryption more difficult, even if the source itself does not encrypt.

timestamp: 32 bits
The timestamp is incremented with a clock frequency determined by the format of data carried as payload. For example, for fixed-rate audio, the timestamp would likely increment by one for each sample. The clock frequency is determined statically by a profile for a set of payload types, during a session through RTCP FMT packets or through other, non-RTP means.

Several consecutive RTP packets may have equal timestamps if they are (logically) generated at once, e.g., belong to the same video frame. The initial value of the timestamp is random, as for the sequence number.

SSRC: 32 bits
Synchronization source identifier. This value is chosen randomly, with the intent that no two synchronization sources within the same channel have the same SSRC value. Details are described in Section 5.2.

CSRC: up to 15 items, 32 bits each
Zero or more content source identifiers. The number of content source identifiers is given by CC. There can be no more than 15 content sources. CSRC identifiers are inserted only by bridges, using the SSRC identifiers of contributing sources. For example, for audio packets, all sources that were mixed together to create a packet are enumerated, allowing correct talker indication at the receiver. A CC value of 0 indicates that the SSRC is the content source. If CC is non-zero, the SSRC (a bridge) is not a content source; a bridge that is also a content source for some packet must explicitly include itself in the CSRC list for that packet. The CSRC list is not modified by translators.

5.2 SSRC Random Identifier Allocation

The SSRC identifier described above is a random 32-bit quantity that is intended to be globally unique within a media session. An example of how to generate such an identifier is presented in Section A.3.

If a source discovers at any time that another source is already using the same SSRC identifier, it randomly chooses a different SSRC identifier. If a source has transmitted packets with the colliding identifier, it should send a BYE control packet with the old SSRC identifier before switching to allow applications to clear any records for this SSRC.

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 [6, p. 33] as

$$1 - \exp(-N^2/2^{L+1}).$$

For $N = 1000$, the probability is roughly 0.01%.

Because the random identifiers are globally unique, they can be used to detect loops that may be introduced by bridges. For each CSRC, the application should check that packets contain a single SSRC value. However, duplicate SSRC values may also indicate a collision resolution in progress.

5.3 RTP Header Extension

A mechanism is provided for extending the RTP data packet header in an application-specific or profile-specific way. If an extension is needed across all profiles, a new version of RTP should be defined instead to make a permanent change to the fixed header.

If the X bit in the RTP header is one, the RTP header (including any CSRC list) is followed by a variable-length header extension. The header extension contains a 16-bit length field that counts the number of octets in the extension, including the four-octet extension header. The first 16 bits of the header extension is defined by the profile.

```
 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              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The main RTP specification does not define any header extensions. Every conformant RTP application needs to be able to skip, but not process the header extension.

6 RTP Control Protocol — RTCP

6.1 Introduction

The RTP control protocol (RTCP) provides two functions: (1) monitoring the distribution of data, and (2) conveying minimal session information.

The first function is performed by the RTCP sender or receiver report packets, described below. This function is an integral part of the RTP's role as a transport protocol, and is mandatory when RTP is used in the IP multicast environment.

The second RTCP function provides support for "loosely controlled" sessions, i.e., where participants enter and leave without membership control and parameter negotiation.

RTCP packets are sent to all members of a session, using the same distribution mechanism as for data packets. The underlying protocol must provide multiplexing of the data and control packets, for example using separate port numbers with UDP. The period between RTCP packets should be varied randomly to avoid synchronization of all sources. Its mean should increase with the number of participants in the session to limit the growth of the overall network and host interrupt load to a small fraction of the load induced by the media data. An algorithm for calculating the period is given in Appendix A.4.

The length of the RTCP period determines, for example, how long a receiver joining a session has to wait until it can identify the source. A receiver may remove from its list of active sites a site that it has not been heard from for a given time-out period; the time-out period may depend on the number of sites or the observed average interarrival time of RTCP messages. A small multiple of the RTCP period is suggested to allow for packet loss. Not every RTCP packet has to contain all RTCP descriptions for a source; for example, SDES EMAIL might only be sent every few messages.

6.2 RTCP packet format

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 but always end on a 32-bit boundary.

The length field and alignment requirement are included to make RTCP packets "stackable". Multiple RTCP packets may be sent in a single packet of the lower layer protocol such as UDP to combine as much information as possible into one packet, particularly for translators and bridges. This is advisable since per-packet processing overhead in the network and in many operating systems is high. For example, in a Unix operating system running the X windowing system, each packet is likely to cause a hardware interrupt, a software interrupt, a context switch and an X event.

Any combination of RTCP packets may be stacked in one lower-layer packet, and each RTCP packet is processed independently. An application may skip RTCP packets with types unknown to it. Additional RTCP packet types may be registered with the Internet Assigned Numbers Authority.

The first RTCP packet is always a report packet, which may be in either of two forms: a sender report (SR) for source that have recently transmitted RTP data packets or receiver reports (RR) for sources that have not recently sent RTP data. It may optionally be followed by more receiver report (RR) packets if the number of sources being reported exceeds 31, the number that will fit into one SR or RR packet. These one or more report packets are followed by an SDES packet containing at least the CNAME item. Finally, FMT, APP, BYE or other, yet to be defined packet types may follow in any order. Packet types may appear more than once.

6.3 SR: Sender report

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|   RC    |  PT=RTCP_SR   |             length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SSRC                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             NTP timestamp, most significant word              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            NTP timestamp, least significant word              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          RTP timestamp                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     sender's packet count                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      sender's byte count                      |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                             SSRC_1                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             cumulative number of packets received             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             cumulative number of packets expected             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    interarrival jitter                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         last SR (LSR)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   delay since last SR (DLSR)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             SSRC_2                            |
                                ...
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                   application-specific extensions             |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The sender report packet consists of two sections. The first section, the actual sender report, is 24 octets long and is present in every sender report packet. The second section contains zero or more reception reports depending on the number of sources heard since the last report. The fields have the following meaning:

type (T): 2 bits
The current value of the type identifier is 2 (two), as in RTP packets.

padding (P): 1 bit
If the padding bit is one, the packet contains some additional octets at the end which are not part of the payload. The very last octet of the packet is a count of how many padding octets

should be ignored. Padding may be needed by some encryption algorithms with fixed block sizes.

reception report count (RC): 5 bits
This field contains the number of reception report blocks contained in this packet. A value of zero is valid.

packet type (PT): 8 bits
The value of the packet type identifier is the constant RTCP_SR, defined in appendixA.

length: 16 bits
The length of this RTCP packet in 32-bit words minus one, including the header and any padding.[3]

SSRC: 32 bits
Synchronization source identifier for the sender of this packet.

NTP timestamp: 64 bits
The NTP timestamp corresponds to the wallclock time when this traffic report is 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 transmitted as it is usually difficult to estimate with any degree of reliability. A sender that can keep track of real time but has no notion of wallclock time may use the elapsed time of the session instead. It is permissible to use the sampling clock to estimate elapsed wallclock time. This is assumed to be less than 68 years, so the high bit will be zero. A sender that has no notion of real time may set the NTP timestamp to zero.

RTP timestamp: 32 bits
Reference timestamp that corresponds to the same time as the NTP timestamp (above). 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.

sender's packet count: 32 bits
Counts the total number of RTP packets transmitted by the source since the source has started transmission and until the time this SR packet was generated.

sender's octet count: 32 bits
Counts the total number of octets transmitted in RTP packets by the source since the source has started transmission and until the time this sender report packet was generated. The octet count includes only the payload of RTP data packets. This field can be used to estimate the overall payload data rate.

Each reception report in the second section of the sender packet conveys statistics on the reception of RTP packets from a single synchronization source. These statistics are:

[3] The offset of one makes zero a valid length and avoids possible infinite loops.

source identifier: 32 bits
Identifies the source to which the information in this report pertains.

cumulative number of packets received: 32 bits
The field contains the total number of RTP packets received from the source since the beginning of reception. By taking the difference in this number between two reception reports from a given source, and dividing by the interval between those two reports, a received packet rate may be calculated.

cumulative number of packets expected: 32 bits
The field contains the total number of packets expected by the receiver, which may be computed according to the algorithm in Appendix A.5. Together with the cumulative number of packets received, a monitor can measure the packet loss rate over both short and long time periods. 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 different significance than 200 out of 1000.) There will be no loss indication (and likely no reception report issued) for a source if all recent packets from that source have been lost.

interarrival jitter: 32 bits
The interarrival jitter is computed by the following algorithm: TBD.

last SR (LSR): 32 bits
The middle 32 bits of the last NTP timestamp received as part of the RTCP reception report packet from the source being reported.

delay since last SR (DLSR): 32 bits
Delay, expressed in units of 1/65536 seconds, between receiving the sender's SR packet and sending this SR packet. The 'last SR' and 'delay since last SR' fields allow the computation of round trip time by the sender of the SR. This may be used to cluster nodes according to propagation delay. If the reception report for SSRC S from receiver R arrives at time A at S, S can compute the round-trip time to R as A − LSR − DLSR. Round-trip may be of limited use for many real-time applications and that some links have very asymmetric delays.

All reported numbers except interarrival jitter are cumulative. The difference between two reports can be used to estimate recent quality of the distribution. A fixed clock (NTP timestamp) is chosen so that quality monitors do not have to be cognizant of the clock rate for the current encoding. If a source cannot compute a particular value, it inserts a value of zero.

A receiver (end system or bridge) should send sender/receiver report packets including a reception report for each source from which it has received RTP packets since the last report, or for as many such sources as will fit. A bridge should not send reception reports on one side for sources it has received on the other side.

A profile may define application specific extensions to the sender report if there is additional information that should be reported regularly about the sender or receivers. If information about receivers is to be included, that data may be structured as an array of blocks parallel to the array of receiver reports in the second section of the sender report.

6.4 RR: Receiver report

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|   RC    |  PT=RTCP_RR   |            length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             SSRC                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                            SSRC_1                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             cumulative number of packets received             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             cumulative number of packets expected             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    interarrival jitter                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          last SR (LSR)                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  delay since last SR (DLSR)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            SSRC_2                             |
                                ...
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                  application-specific extensions              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The RR packet is issued in place of an SR packet only if the application has not recently sent any RTP data packets. (Unless specified by a profile, the timeout delay between sending the last RTP packet and ceasing to send SR packets should be a small multiple of the current reporting interval.) The packet fields have the same meaning as for the SR packet. Additional RR packets may follow the initial SR or RR packet if there are more than 31 sources to be reported.

6.5 SDES: Source description

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|    CC   | PT=RTCP_SDES  |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          SSRC/CSRC_1                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          SDES items                           |
|                              ...                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                          SSRC/CSRC_2                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          SDES items                           |
|                              ...                              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The SDES packet is composed of a header and zero or more chunks containing items describing the sources identified in those chunks. The items are described individually below.

type (T), padding (P), payload type (SDES), length:
As described for the SR packet.

CC: 5 bits
This field contains the number of SSRC/CSRC chunks included in this SDES packet.

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 (including this two-octet header) and text. Items are contiguous, i.e., items are not individually padded to a 32-bit boundary. Text is not zero terminated. The list of items in each chunk is terminated by one or more zeroes to denote the end of the list and pad until the next 32-bit boundary. An SDES packet with zero chunks or a chunk with zero items 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 bridge sends one SDES packet containing a chunk for each content source from which it is receiving SDES information, or more than one SDES packet if there are more than 31 such sources.

The following SDES items are currently defined. Additional items may be defined in a profile; some items shown here may be useful for particular profiles only. Not all items need to be sent with every SDES packet, except for the CNAME item, which is mandatory.[4]

[4]Items are defined here rather than in the profile to simplify profile-independent applications, using common type numbers.

6.5.1 CNAME: Canonical end-point identifier

```
 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     |     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 to an identifier for the source that remains constant.
- Like the SSRC identifier, the CNAME identifier should also be unique within one medium of a session.
- To provide a binding among multiple media tools used in a session by one participant, 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", where "host" is the fully qualified domain name of the host from which the real-time data originates, formatted according to the rules specified in RFC 1034, RFC 1035 and Section 2.1 of RFC 1123. The "host" form may be used if a user name is not available, for example on single-user systems. Only if a system cannot obtain a valid domain name, it may use the printable representation of its lowest numbered numeric network address. Hosts using IP Version 4 use the 'dotted decimal' (also known as 'dotted quad') representation. Application writers should be aware that address assignments such as the Net-10 assignment proposed in RFC 1597 may create IP network addresses that are not globally unique. This may create difficulties if sites that do not have direct IP connectivity to the public Internet forward RTP packets to the public Internet through an RTP-level firewall. (See also RFC 1627.) To handle this case, applications should provide a means to configure a unique name.

Examples are:

"doe@sleepy.megacorp.com" or "sleepy.megacorp.com" or "doe@192.35.149.160" or "192.35.149.160"

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 real-life name. The host name is not necessarily identical to the electronic mail address of the participant.

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.

6.5.2 NAME: User name

```
 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      |     length    | common name of source       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The real name used to describe the source, e.g., "John Doe, Bit Recycler, Megacorp". This name 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 (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 across the session.

6.5.3 EMAIL: User's electronic mail address

```
 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      |     length    | email address of source     ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

6.5.4 LOC: Geographic user location

```
 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      |     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.

6.5.5 TXT: Text describing the source

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      TXT      |     length    | text describing source       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Message describing the current state of the source, e.g., "can't talk, having lunch". During a seminar, this field might be used to convey the title of the talk. The TXT value is likely to change during a session.

6.6 BYE: Goodbye

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|    CC   | PT=RTCP_BYE |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           SSRC/CSRC                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       reason for leaving                     ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

type (T), padding (P), payload type (BYE), length:
As described for the SR packet.

CC: 5 bits
This field contains the number of SSRC/CSRC identifiers included in this SDES packet. A count value of zero is valid, but meaningless.

If a BYE packet is received by a bridge, the bridge forwards the BYE packet with the SSRC/CSRCS identifier(s) unchanged. If a bridge shuts down, it should send a BYE packet listing all content sources it handles, as well as its own SSRC identifier. Optionally, the BYE packet may include a string indicating the reason for leaving, e.g., "equipment malfunction".

6.7 FMT: Payload type mapping

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P|reserved |    PT=FMT     |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SSRC                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 reserved                      |   RTP-PT      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    nominal clock frequency, integer part      |   fraction    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           format name                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     format-dependent data                     |
|                               ...                             |
```

A FMT mapping establishes the interpretation of a given format value carried in the fixed RTP header starting at the packet containing the FMT option. The new interpretation applies only to packets from the same synchronization source as the packet containing the FMT option, i.e., it is source-specific. An existing mapping, e.g., one established through a profile or directory service, must not be changed. Sources should not send FMT packets for encodings that are defined by a profile.

type (T), padding (P), payload type (PT), length:
See description of SR packet.

SSRC: 32 bits
Synchronization source identifier for the sender of this packet.

RTP payload type (RTP-PT): 8 bits
The payload type field corresponds to the index value from the payload type field in the RTP header. This mapping is expected to remain constant for the duration of a session.

nominal clock frequency: 32 bits
The nominal clock rate is specified as the closest fixed point value consisting of a 24 bit integer and an 8 bit binary fraction. A value of zero indicates that the clock rate is unknown or variable. The actual clock rate of a source may well differ slightly from this nominal value. For example, an audio source might indicate a nominal clock rate of 8000 Hz, while it emits samples at an actual rate of 8005 Hz, due to imperfections in the local crystal oscillator. The nominal clock rate is used since measuring the actual clock rate may be difficult. Furthermore, for precise synchronization over longer time periods, the SR clock values are to be used. The clock rate information allows recorders to play back RTP packets without knowledge of the semantics of the payload data.

format name: 32 bits
The format name describes the format in an unambiguous way and is registered with the

Internet Assigned Numbers Authority. The format name is interpreted as a sequence of four ASCII characters, with uppercase and lowercase characters treated as distinct. Format names beginning with the letter 'X' are reserved for experimental use and not subject to registration.

format-dependent data: variable length
Format-dependent data may or may not appear in a FMT packet. It is interpreted by the application and not RTCP itself.

6.8 APP: Application-defined

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=2|P| subtype |    PT=APP     |             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.

type (T), padding (P), packet type (APP), length:
As defined for the SR packet.

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.

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.

7 Security

7.1 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. For example, 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 [7]. Therefore, the use of security mechanisms with RTP is important.

As a first step, RTCP makes it easy for all participants in a session to identify themselves; if deemed important for a particular application, it is the responsibility of the application writer to make listening without identification difficult. It should be noted, however, that privacy of the payload can generally be assured only by encryption.

The security measures described below can be used to implement confidentiality. Authentication and message integrity are not defined in the current specificiation of RTP. Security services might also be provided at the IP layer as security mechanisms are developed for that layer.

The periodic transmission of RTCP or empty RTP packets from sources that are otherwise idle may make it possible to detect denial-of-service attacks, as the receiver can detect the absence of these expected messages. The messages that are received must be verified for integrity and authenticated before being accepted for this purpose.

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

The section below defines a confidentiality security service and defines standard algorithms for both RTP and RTCP. Other services, other implementations of services and other algorithms may be defined in the future. 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 specifies which of the services and algorithms should be offered by applications, and may provide guidance as to their appropriate use.

7.2 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.

All RTP and RTCP packets in a single lower-layer protocol data unit are encrypted as a unit. For RTCP, it is allowed to send some such lower-layer packets encrypted, others in the clear. (This accomodates monitors that are not privy to the encryption key.) For RTP, no additional data structures are required. For RTCP, a 32-bit random number is prepended to the unit before encryption to deter known plaintext attacks. The presence of encryption and the use of the correct key are confirmed by the receiver through header or payload consistency checks. An example of such a consistency check is given in Section A.1.

The default encryption algorithm is the Data Encryption Standard (DES) algorithm in CBC (cipher block chaining) mode, as described in Section 1.1 of RFC 1423 [8], 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 RTCP packets. For details on the use of CBC initialization vectors, see [9]. Implementations that support encryption should always support the DES algorithm in CBC mode.

As an alternative to encryption 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.

8 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 specifications.

RTP relies on the underlying protocol(s) to provide demultiplexing. For UDP and similar protocols, RTCP always uses the port number of the corresponding RTP stream plus one, unless specified otherwise by a protocol.

RTP 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 transport mechanism.

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. TCP is an example of such a protocol. 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.

9 RTP Profiles

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 defining a *profile* to select the appropriate choices for a particular class of applications and environment. A profile for audio and video applications may be found in the companion Internet draft `draft-ietf-avt-profile`.

Within this specification, the following possible uses of a profile have been identified, but this list is not meant to be exclusive:

- Define a set of formats (e.g., media encodings) and a default mapping of those formats to payload type values.
- Define the number and interpretation of the RTP marker bits, if different from the default specified in Section 5.1.
- Define new application-class-specific RTCP packets, or the data format, preferred use, or required use of particular RTCP packets.
- Specify that a particular underlying network or transport layer protocol will be used to carry RTP packets.
- Specify the mapping of RTP and RTCP to transport-level names, e.g., UDP ports, if different from the mapping defined in Section 8.
- Specify encapsulation of RTP packets that are to be used always or with particular underlying protocols.

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. For example, additional RTCP packet types or formats can be defined and registered through the Internet Assigned Numbers Authority for publication in the Assigned Numbers RFC as an alternative to publishing a new profile specification.

A Implementation Notes

We describe aspects of the receiver implementation in this section. 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.

The following definitions are used for all examples; the structure definitions are valid for 32-bit big-endian architectures only. Bit fields are assumed to be packed tightly, with no additional padding.

```
#include <sys/types.h>

/* the definitions below are valid for 32-bit architectures and will
   have to be adjusted for 16- or 64-bit architectures */
typedef u_char  u_int8;
typedef u_long  u_int32;
typedef u_short u_int16;


/*
rtp.h  --  RTP header file
*/

#include <types.h>

#define RTP_SEQ_MOD (1<<16)
#define RTP_TS_MOD  (0xffffffff)

#define RTP_MAX_SDES 256   /* maximum text length for SDES */

typedef enum {
  RTCP_SR,
  RTCP_RR,
  RTCP_SDES,
  RTCP_BYE,
  RTCP_FMT,
  RTCP_APP
} rtcp_type_t;

typedef enum {
  RTCP_SDES_CNAME,
  RTCP_SDES_NAME,
  RTCP_SDES_EMAIL,
  RTCP_SDES_LOC,
```

```
  RTCP_SDES_TXT,
} rtcp_sdes_type_t;

typedef struct {
  unsigned int type:2;      /* packet type */
  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 */
  u_int16 seq;              /* sequence number */
  u_int32 ts;               /* timestamp */
  u_int32 ssrc;             /* synchronization source */
  u_int32 csrc[1];          /* optional CSRC list */
} rtp_hdr_t;

typedef struct {
  unsigned int type:2;      /* packet type */
  unsigned int p:1;         /* padding flag */
  unsigned int count:5;     /* varies by payload type */
  unsigned int pt:8;        /* payload type */
  u_int16 length;           /* packet length */
} rtcp_common_t;

/* reception report */
typedef struct {
  u_int32 received;         /* cumulative number of packets received */
  u_int32 expected;         /* cumulative number of packets expected */
  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;

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

/* one RTCP packet */
typedef struct {
  rtcp_common_t common;     /* common header */
  union {
    /* sender report (SR) */
    struct {
```

```
      u_int32 rtp_ts;      /* RTP timestamp */
      u_int32 ssrc;        /* source this RTCP packet refers to */
      u_int32 ntp_sec;     /* NTP timestamp */
      u_int32 ntp_frac;
      /* variable-length list */
      rtcp_rr_t rr[1];
    } sr;

    /* reception report (RR) */
    struct {
      u_int32 rtp_ts;      /* RTP timestamp */
      u_int32 ssrc;        /* source this RTCP packet refers to */
      /* variable-length list */
      rtcp_rr_t rr[1];
    } rr;

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

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

    /* format (FMT) */
    struct {
      u_int32 src;         /* SSRC */
      u_int32 freq;        /* clock frequency */
      char name[4];        /* format name */
    } fmt;
  } r;
} rtcp_t;
```

A.1 RTP Header Consistency Check

The following checks may be used to determine whether an RTP header is likely to be valid, given a previously received RTP packet:

- RTP type field value equal to 2

- payload type defined
- RTP sequence number one higher than previous packet
- if packets contain fixed number of timestamp counts, comparison of timestamp increment with sequence number increment
- length of RTP packet consistent with CC and payload type

Depending on the application, algorithms may exploit additional knowledge, e.g., the expected increment in timestamps between packets. Note that this algorithm is likely to occasionally create false alarms.

A.2 Parsing RTCP Packets

The following code fragment walks through one or more RTCP packets, checking for invalid length fields. It may also be advisable to treat the packet type and payload type as a single field for checking and branching.

```
u_int32 len;       /* length of combined RTCP packets in words */
rtcp_t *r;         /* RTCP header */

while (len > 0) {
  len -= r->common.length + 1;
  if (len < 0) {
    /* something wrong with packet format */
    break;
  }
  switch (r->common.pt) {
  case RTCP_SR:
    break;
  default:
    /* invalid type */
    break;
  }
  r = (rtcp_t *)((u_int32 *)r + r->common.length + 1);
}
```

A.3 Generating a Random 32-bit Identifier

The following subroutine generates a random 32-bit identifier using the MD5 routines published in RFC 1321. 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()`. “Live” video or audio samples are also a good source of random numbers, but care must be taken to avoid that a turned-off microphone or blinded camera is used as a source.

```
/*
* 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 "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.
*/
u_long random32(void)
```

```
{
  struct {
    struct timeval tv;
    pid_t pid;
    u_long hostid;
    uid_t  uid;
    gid_t  gid;
    char   name[8];
  } s;

  gettimeofday(&s.tv, 0);
  s.pid    = getpid();
  s.hostid = gethostid();
  s.uid    = getuid();
  s.gid    = getgid();
  gethostname(s.name, sizeof(s.name));

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

A.4 Estimating the Number of Participants and Computing the RTCP Transmission Period

This algorithm has been designed and implemented, but the writeup is not yet available. Details are to be supplied in the next draft. Two important characteristics are that it starts with a conservatively large RTCP interval and converges quickly to the desired value.

A.5 Determining the Expected Number of RTP Packets

In order to compute packet loss rates, the number of packets expected and actually received needs to be known. The number of packets expected can be computed by the receiver by tracking the first sequence number received (`seq0`), the last sequence number received, `seq`, and the number of complete sequence number cycles:

```
expected = cycles * 65536 + seq - seq0 + 1;
```

The cycle count `cycles` is updated for each packet, where `seq_prior` is the sequence number of the prior packet. The cycle count is incremented when the sequence number wraps around in the "forward" direction, and needs to be decremented if the sequence number wraps around in the "backward" direction.

```
unsigned short seq, seq_prior;

if (seq > seq_prior) {
  if (seq - seq_prior > 32768) {
    /* out-of-order packet with wrap-around (e.g., 65530 preceded by 3) */
    cycles--;
  }
}
else if (seq < seq_prior) {
  if (seq - seq_prior > 32768) {
    /* out-of-order packet (e.g., 2 preceded by 3) */
  }
  else {
    /* wrap-around (e.g., 3 preceded by 65530) */
    cycles++;
  }
}

seq_prior = seq;
```

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.

B Addresses of Authors

Stephen Casner
University of Southern California/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
United States
telephone: +1 310 822 1511 (extension 153)
electronic mail: casner@isi.edu

Henning Schulzrinne
GMD Fokus
Hardenbergplatz 2

D-10623 Berlin
Germany
telephone: +49 30 25499 219
facsimile: +49 30 25499 202
electronic mail: `hgs@fokus.gmd.de`

Ron Frederick
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
United States
telephone: `+1 415 812 4459`
electronic mail: `frederic@parc.xerox.com`

Van Jacobson
Lawrence Berkeley Laboratory
United States
telephone:
electronic mail: `van@ee.lbl.gov`

References

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

[2] J. Postel, "Internet protocol," Request for Comments (Standard) RFC 791, Internet Engineering Task Force, Sept. 1981. Obsoletes RFC0760.

[3] International Standards Organization, "ISO/IEC DIS 10646-1:1993 information technology – universal multiple-octet coded character set (UCS) – part I: Architecture and basic multilingual plane," 1993.

[4] The Unicode Consortium, *The Unicode Standard*. New York, New York: Addison-Wesley, 1991.

[5] D. Mills, "Network time protocol (v3)," Request for Comments (Proposed Standard) RFC 1305, Internet Engineering Task Force, Apr. 1992. Obsoletes RFC1119.

[6] 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.

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

[8] D. Balenson, "Privacy enhancement for internet electronic mail: Part III: algorithms, modes, and identifiers," Request for Comments (Proposed Standard) RFC 1423, Internet Engineering Task Force, Feb. 1993. Obsoletes RFC1115.

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