INTERNATIONAL TELECOMMUNICATION UNION

ITU-T J.121

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(02/2002)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Interactive systems for digital television distribution

Quality control protocol for webcasting

ITU-T Recommendation J.121

ITU-T J-SERIES RECOMMENDATIONS

CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

General Recommendations	J.1–J.9
General specifications for analogue sound-programme transmission	J.10–J.19
Performance characteristics of analogue sound-programme circuits	J.20–J.29
Equipment and lines used for analogue sound-programme circuits	J.30–J.39
Digital encoders for analogue sound-programme signals	J.40–J.49
Digital transmission of sound-programme signals	J.50–J.59
Circuits for analogue television transmission	J.60–J.69
Analogue television transmission over metallic lines and interconnection with radio-relay links	J.70–J.79
Digital transmission of television signals	J.80–J.89
Ancillary digital services for television transmission	J.90–J.99
Operational requirements and methods for television transmission	J.100–J.109
Interactive systems for digital television distribution	**J.110–J.129**
Transport of MPEG-2 signals on packetised networks	J.130–J.139
Measurement of the quality of service	J.140–J.149
Digital television distribution through local subscriber networks	J.150–J.159
IPCablecom	J.160–J.179
Miscellaneous	J.180–J.199
Application for Interactive Digital Television	J.200–J.209

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation J.121

Quality control protocol for webcasting

Summary

This Recommendation defines protocols between a server and a client in order to distribute sound and television programs by J.120, i.e. perform "webcasting", over a general IP network, which is a non-QoS guaranteed network where data error or packet losses may occur. The use of these protocols results in an improvement of quality.

A scheme is provided wherein a quality report is sent from a client to the server, and the server performs optimum data distributions according to the client's report.

This Recommendation defines the kinds of parameters that a client should report to the server, as well as a transmission protocol that is used for sending the report from the client to the server. This Recommendation also defines transmission of supporting parameters that are useful for the server to analyze the client's report.

Source

ITU-T Recommendation J.121 was prepared by ITU-T Study Group 9 (2001-2004) and approved under the WTSA Resolution 1 procedure on 13 February 2002.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

ITU-T Recommendation J.121

Quality control protocol for webcasting

1 Scope

The response of the server after receiving the client's report is outside the scope of this Recommendation.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

2.1 Normative references

– ITU-T Recommendation H.225.0 (2000), *Call signalling protocols and media stream packetization for packet-based multimedia communication systems*.

– ITU-T Recommendation J.120 (2000), *Distribution of sound and television programs over the IP network.*

2.2 Informative references

– IETF RFC 1889, (1996), *RTP: A Transport Protocol for Real-Time Applications*.

3 Terms and definitions

This Recommendation defines the following terms:

3.1 real-time transport protocol (RTP): A transport protocol for real-time applications defined in ITU-T Rec. H.225.0.

3.2 RTP control protocol (RTCP): A control protocol for RTP packets defined in ITU-T Rec. H.225.0.

3.3 webcasting: "Webcasting" is defined in ITU-T Rec. J.120. Distribution of sound and television programs over the IP network.

4 Abbreviations and conventions

4.1 Abbreviations

This Recommendation uses the following abbreviations:

IP Internet Protocol

RTCP RTP Control Protocol

RTP Real-time Transport Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

4.2 Conventions

If this Recommendation is implemented, the key words "MUST" and "SHALL" as well as "REQUIRED" are to be interpreted as indicating a mandatory aspect of this specification.

The keywords indicating a certain level of significance of a particular requirement that are used throughout this Recommendation are summarized below.

"MUST" This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.

"MUST NOT" This phrase means that the item is an absolute prohibition of this specification.

"SHOULD" This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

"SHOULD NOT" This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

"MAY" This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

5 System definition

Figure 1 shows the system and its signal flow between the server and the clients. In this scheme, Sender Report (SR) and Receiver Report (RR) of RTCP are used for exchanging information between the server and the client. The RTCP is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the media data packets. The underlying protocol shall provide multiplexing of the data and control packets, for example using separate port numbers with UDP.

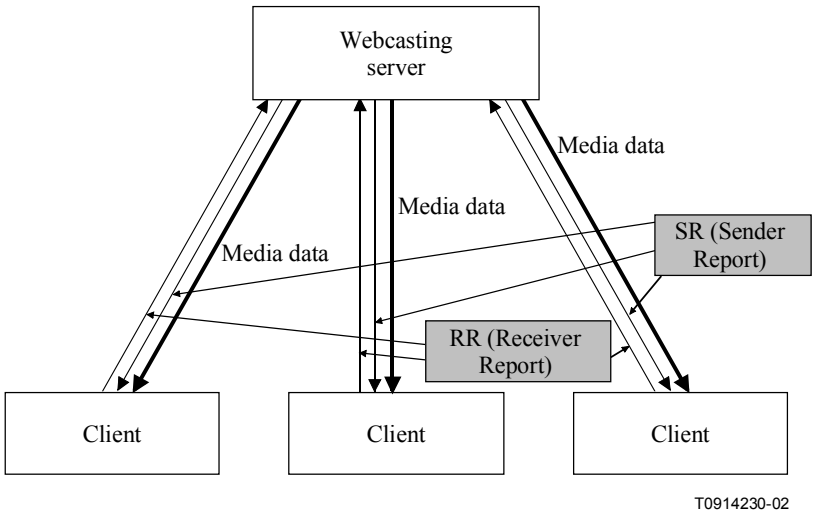

Figure 1/J.121 – Block diagram of quality control scheme for webcasting

5.1 Procedure for the server

5.1.1 Transmission of measurement support information

First of all, the server transmits measurement support information which includes a reference time and information regarding the transmitted packets to the client. This measurement information is used by the client to make a client's report and used by the server to analyze the client's report. Although the transmission interval for this information is not defined by this Recommendation, clause 7 describes recommended transmission interval. This information is conveyed by a SR (Sender Report) packet defined in 6.1. The SR packet is sent using a separate port number from the media data (video/audio). The port number is given by the session initiation of the J.120 protocol, which is just after (plus one) the Media Data RTP port number.

5.1.2 Reception of the client's report

The server shall always be in such a state as to be able to receive the report packet from the client.

5.1.3 Distribution of media data

The server should perform optimum distribution of media data (audio/video) according to the results of analyzing the report packet from the client. The actual method of controlling distribution is a matter of implementation, and is outside the scope of this Recommendation.

5.2 Procedure for the client

5.2.1 Monitoring of transmission quality

The client monitors the condition under which the media data is received and measures the transmission quality. The parameters to be measured are described in 6.2. In the measurement, the client shall calculate these parameters based on the aforesaid measurement support information transmitted from the server.

5.2.2 Transmission of the client's report to the server

The transmission quality measured at the client is transmitted to the server as a client's report. The client's report shall be conveyed as an RR (Receiver Report) packet defined in 6.2. The RR packet is sent using given port number by the session initiation of the J.120 protocol.

6 Definition of a Report Packet

6.1 Sender Report (SR)

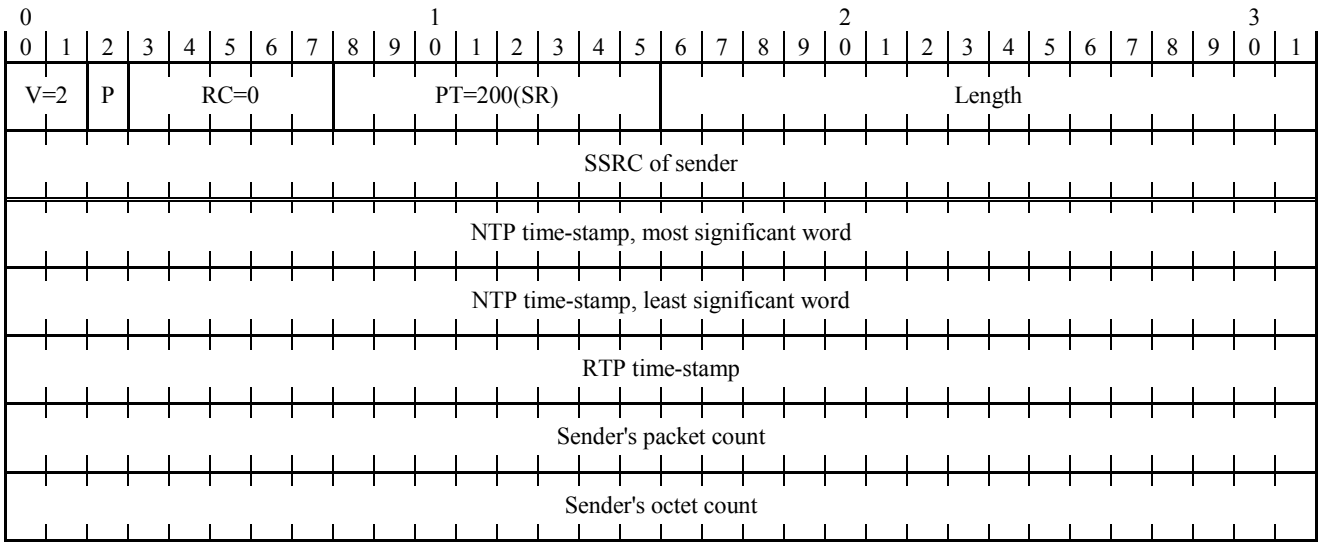

Figure 2/J.121 – Packet format of the Sender Report (SR)

The sender report packet consists of two sections. 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 Recommendation is two (2).

padding (P): 1 bit. If the padding bit is set, this RTCP packet contains some additional padding octets at the end which are not part of the control information. The last octet of the padding is a count of how many padding octets should be ignored. Padding may be needed by some encryption algorithms with fixed block sizes. In a compound RTCP packet, padding should only be required on the last individual packet because the compound packet is encrypted as a whole.

reception report count (RC): 5 bits. The number of reception report blocks contained in this packet. A value is always zero.

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 time-stamp: 64 bits. Indicates the wallclock time when this report was sent so that it may be used in combination with time-stamps returned in reception reports from other receivers to measure round-trip propagation to those receivers. Receivers should expect that the measurement accuracy of the time-stamp may be limited to far less than the resolution of the NTP time-stamp. The measurement uncertainty of the time-stamp is not indicated as it may not be known. A sender that can keep track of elapsed time but has no notion of wallclock time may use the elapsed time since joining the session instead. This is assumed to be less than 68 years, so the high bit will be zero. It is

permissible to use the sampling clock to estimate elapsed wallclock time. A sender that has no notion of wallclock or elapsed time may set the NTP time-stamp to zero.

RTP time-stamp: 32 bits. Corresponds to the same time as the NTP time-stamp (above), but in the same units and with the same random offset as the RTP time-stamps in data packets. This correspondence may be used for intra- and inter-media synchronization for sources whose NTP time-stamps are synchronized, and may be used by media-independent receivers to estimate the nominal RTP clock frequency. Note that in most cases this time-stamp will not be equal to the RTP time-stamp in any adjacent data packet. Rather, it is calculated from the corresponding NTP time-stamp using the relationship between the RTP time-stamp 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.

6.2 Receiver Report (RR)

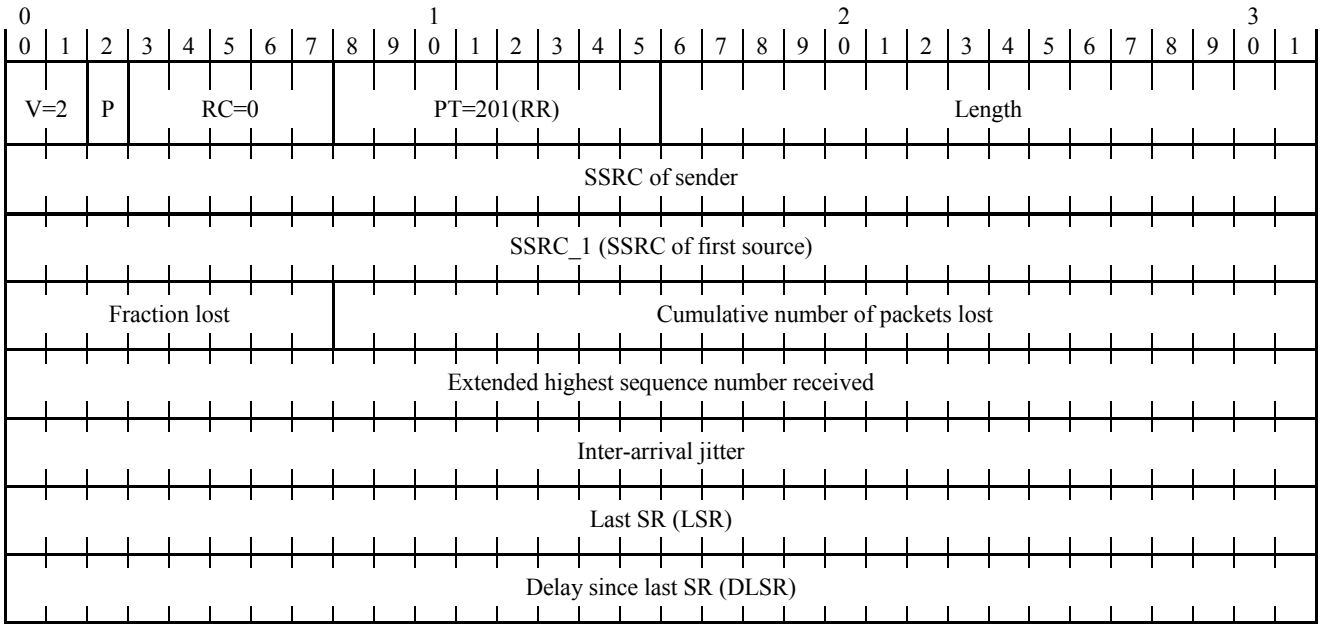

Figure 3/J.121 – Packet format of Receiver Report (RR)

The receiver report packet consists of two sections. The first section, the header which is 8 octets long, is the same as that of the SR packet except that the packet type field contains the constant 201. The fields is defined in 6.1.

The second section contains zero or more reception report blocks depending on the number of other sources heard by this receiver 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. 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 III.

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 II. Note that different receivers within the same session will generate different extensions to the sequence number if their start times differ significantly.

Inter-arrival jitter: 32 bits. An estimate of the statistical variance of the RTP data packet inter-arrival time, measured in time-stamp units and expressed as an unsigned integer. The inter-arrival 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 time-stamp and the receiver's clock at the time of arrival, measured in the same units.

If *Si* is the RTP time-stamp from packet *i*, and *Ri* is the time of arrival in RTP time-stamp units for packet *i*, then for two packets *i* and *j*, *D* may be expressed as:

$$D(i+j) = (Rj - Ri) - (Sj - Si) = (Rj - Sj) - (Ri - Si)$$

The inter-arrival 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:

$$J = J + \frac{|D(i-1,i)| - J}{16}$$

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. A sample implementation is shown in Appendix V.

Last SR time-stamp (LSR): 32 bits. The middle 32 bits out of 64 in the NTP time-stamp (as explained in A.4/H.225.0: Byte order, alignment, and time format) 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 time-stamp (LSR) field, and then subtracting this field to leave the round-trip propagation delay as (A – LSR – DLSR). This is illustrated in Figure 4. This may be used as an approximate measure of distance to cluster receivers, although some links have very asymmetric delays.

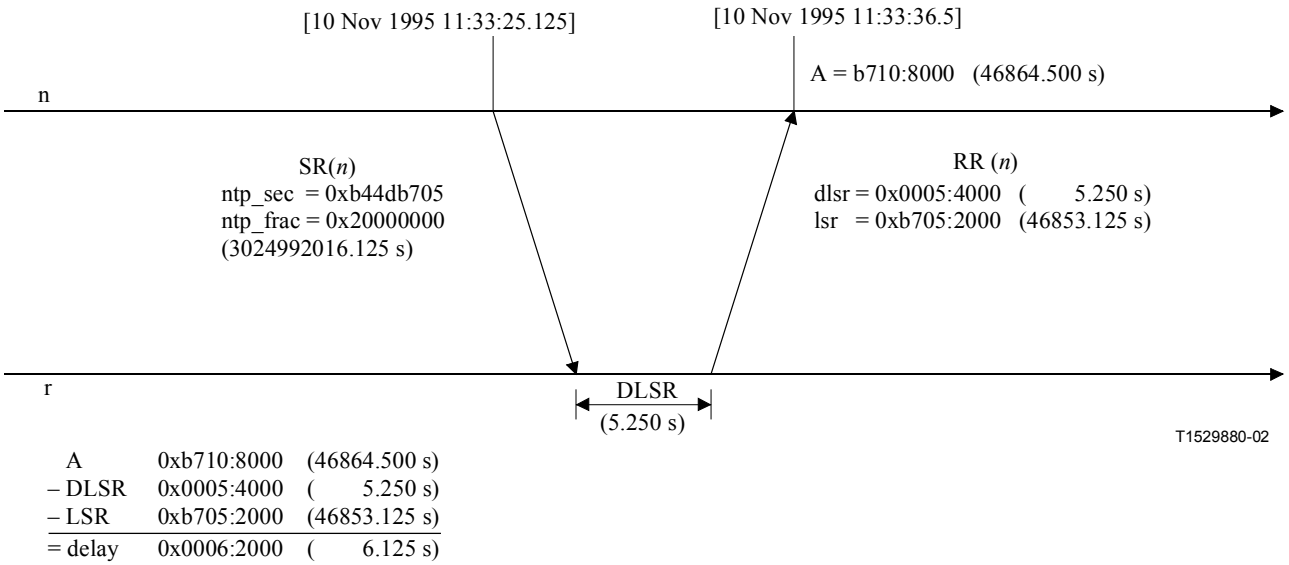

Figure 4/J.121 – Example for round-trip time computation

7 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.

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, or it might just be a reasonable share. The session bandwidth may be chosen based on 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. 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 suggested that the fraction of the session bandwidth allocated to RTCP be fixed at 5%. While the value of this and other constants in the interval calculation is 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.

The algorithm described in Appendix IV is 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:

- Senders are collectively allocated at least 1/4 of the control traffic bandwidth 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.
- The calculated interval between RTCP packets is required to be greater than a minimum of 5 seconds to avoid having bursts of RTCP packets exceed the allowed bandwidth when the number of participants is small and the traffic is not smoothed according to the law of large numbers.
- 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. The first RTCP packet sent after joining a session is also delayed by a random variation of half the minimum RTCP interval in case the application is started at multiple sites simultaneously, for example as initiated by a session announcement.
- 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.

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.

Appendix I

Algorithms

This appendix provides 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 */
     u_int16 seq;                      /* sequence number */
     u_int32 ts;                       /* time-stamp */
     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;                    /* inter-arrival 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 time-stamp */
            u_int32 ntp_frac;
            u_int32 rtp_ts;               /* RTP time-stamp */
            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 */
            p_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;
```

Appendix II

RTP 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 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, 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 s->probation marks the source as not yet valid so the state may be discarded after a short timeout rather than a long one.

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 time-stamp increment is the same for all packets, the time-stamp 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.

Appendix III

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 II.

```
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 number is carried in 24 bits, it should be clamped at 0xffffff rather than wrap around to zero.

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.

Appendix IV

Computing the RTCP transmission interval

The following function returns the time between transmissions of RTCP packets, measured in seconds. It should be called after sending one compound RTCP packet to calculate the delay until the next should be sent. This function should also be called to calculate the delay before sending the first RTCP packet upon start-up rather than send the packet immediately. This avoids any burst of RTCP packets if an application is started at many sites simultaneously, for example as a result of a session announcement.

The parameters have the following meaning:

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 start-up.

senders: Number of active senders since sending last report, known from construction of receiver reports for this RTCP packet. Includes ourselves, if we also sent during this interval.

members: The estimated number of session members, including ourselves. Incremented as we discover new session members from the receipt of RTP or RTCP packets, and decremented as session members leave (via RTCP BYE) or their state is timed out (30 minutes is recommended). On the first call, this parameter should have the value 1.

we_sent: Flag that is true if we have sent data during the last two RTCP intervals. If the flag is true, the compound RTCP packet just sent contained an SR packet.

packet_size: The size of the compound RTCP packet just sent, in octets, including the network encapsulation (e.g. 28 octets for UDP over IP).

avg_rtcp_size: Pointer to estimator for compound RTCP packet size; initialized and updated by this function for the packet just sent, and also updated by an identical line of code in the RTCP receive routine for every RTCP packet received from other participants in the session.

initial: Flag that is true for the first call upon start-up to calculate the time until the first report should be sent.

```
#include <math.h>

     double rtcp_interval(int members,
     int senders,
     double rtcp_bw,
     int we_sent,
     int packet_size,
     int *avg_rtcp_size,
     int initial)
     {
     /*
     * Minimum 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);
     /*
     * Gain (smoothing constant) for the low-pass filter that
     * estimates the average RTCP packet size (see Cadzow reference).
     */
     double const RTCP_SIZE_GAIN = (1./16.);

     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.  The average RTCP size is initialized
* to 128 octets which is conservative (it assumes everyone else
* is generating SRs instead of RRs: 20 IP + 8 UDP + 52 SR + 48
* SDES CNAME).
*/
if (initial) {
rtcp_min_time /= 2;
*avg_rtcp_size = 128;
}

/*
* 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;
}
}

/*
* Update the average size estimate by the size of the report
* packet we just sent.
*/
*avg_rtcp_size += (packet_size - *avg_rtcp_size)*RTCP_SIZE_GAIN;

/*
* 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);
}
```

Appendix V

Estimating the inter-arrival jitter

The code fragments below implement the algorithm given in 6.2 for calculating an estimate of the statistical variance of the RTP data inter-arrival time to be inserted in the inter-arrival jitter field of reception reports. The inputs are `r->ts`, the time-stamp 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 time-stamp 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;
```

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2002