RTP Payload Format for JPEG-compressed Video

Status of this Memo

Copyright Notice

Abstract

   This memo describes the RTP payload format for JPEG video streams.
   The packet format is optimized for real-time video streams where
   codec parameters change rarely from frame to frame.

   This document 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 author(s).

Changes from RFC 2035

   Most of this memo is identical to RFC 2035. The changes made to the
   protocol are summarized in Appendix D.

Key Words

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [9].

1. Introduction

   The Joint Photographic Experts Group (JPEG) standard [1,2,3] defines
   a family of compression algorithms for continuous-tone, still images.
   This still image compression standard can be applied to video by
   compressing each frame of video as an independent still image and
   transmitting them in series. Video coded in this fashion is often
   called Motion-JPEG.

   We first give an overview of JPEG and then describe the specific
   subset of JPEG that is supported in RTP and the mechanism by which
   JPEG frames are carried as RTP payloads.

   The JPEG standard defines four modes of operation: the sequential DCT
   mode, the progressive DCT mode, the lossless mode, and the
   hierarchical mode. Depending on the mode, the image is represented
   in one or more passes. Each pass (called a frame in the JPEG
   standard) is further broken down into one or more scans. Within each
   scan, there are one to four components, which represent the three
   components of a color signal (e.g., "red, green, and blue", or a
   luminance signal and two chrominance signals). These components can
   be encoded as separate scans or interleaved into a single scan.

   Each frame and scan is preceded with a header containing optional
   definitions for compression parameters like quantization tables and
   Huffman coding tables. The headers and optional parameters are
   identified with "markers" and comprise a marker segment; each scan
   appears as an entropy-coded bit stream within two marker segments.
   Markers are aligned to byte boundaries and (in general) cannot appear
   in the entropy-coded segment, allowing scan boundaries to be
   determined without parsing the bit stream.

   Compressed data is represented in one of three formats: the
   interchange format, the abbreviated format, or the table-
   specification format. The interchange format contains definitions
   for all the tables used by the entropy-coded segments, while the
   abbreviated format might omit some assuming they were defined out-
   of-band or by a "previous" image.

   The JPEG standard does not define the meaning or format of the
   components that comprise the image. Attributes like the color space
   and pixel aspect ratio must be specified out-of-band with respect to

the JPEG bit stream. The JPEG File Interchange Format (JFIF) [4] is
a de-facto standard that provides this extra information using an
application marker segment (APP0). Note that a JFIF file is simply a
JPEG interchange format image along with the APP0 segment. In the
case of video, additional parameters must be defined out-of-band
(e.g., frame rate, interlaced vs. non-interlaced, etc.).

While the JPEG standard provides a rich set of algorithms for
flexible compression, cost-effective hardware implementations of the
full standard have not appeared. Instead, most hardware JPEG video
codecs implement only a subset of the sequential DCT mode of
operation. Typically, marker segments are interpreted in software
(which "re-programs" the hardware) and the hardware is presented with
a single, interleaved entropy-coded scan represented in the YUV color
space.

The scan contains an ordered sequence of Minimum Coded Units, or
MCUs, which are the smallest group of image data coded in a JPEG bit
stream. Each MCU defines the image data for a small rectangular
block of the output image.

Restart markers in the JPEG data denote a point where the decoder
should reset its state. As defined by JPEG, restart markers are the
only type of marker that may appear embedded in the entropy-coded
segment, and they may only appear on an MCU boundary. A "restart
interval" is defined to be a block of data containing a restart
marker followed by some fixed number of MCUs. An exception is made
for the first restart interval in each frame, which omits the initial
restart marker and just begins with the MCU data. When these markers
are used, each frame is composed of some fixed number of back-to-back
restart intervals.

2. JPEG Over RTP

To maximize interoperability among hardware-based codecs, we assume
the sequential DCT operating mode [1,Annex F] and restrict the set of
predefined RTP/JPEG "type codes" (defined below) to single-scan,
interleaved images. While this is more restrictive than even
baseline JPEG, many hardware implementation fall short of the
baseline specification (e.g., most hardware cannot decode non-
interleaved scans).

In practice, most of the table-specification data rarely changes from
frame to frame within a single video stream. Therefore RTP/JPEG data
is represented in abbreviated format, with all of the tables omitted
from the bit stream where possible. Each frame begins immediately
with the (single) entropy-coded scan. The information that would
otherwise be in both the frame and scan headers is represented

entirely within the RTP/JPEG header (defined below) that lies between
the RTP header and the JPEG payload.

While parameters like Huffman tables and color space are likely to
remain fixed for the lifetime of the video stream, other parameters
should be allowed to vary, notably the quantization tables and image
size (e.g., to implement rate-adaptive transmission or allow a user
to adjust the "quality level" or resolution manually). Thus explicit
fields in the RTP/JPEG header are allocated to represent this
information. Since only a small set of quantization tables are
typically used, we encode the entire set of quantization tables in a
small integer field. Customized quantization tables are accommodated
by using a special range of values in this field, and then placing
the table before the beginning of the JPEG payload. The image width
and height are encoded explicitly.

Because JPEG frames are typically larger than the underlying
network's maximum packet size, frames must often be fragmented into
several packets. One approach is to allow the network layer below
RTP (e.g., IP) to perform the fragmentation. However, this precludes
rate-controlling the resulting packet stream or partial delivery in
the presence of loss, and frames may be larger than the maximum
network layer reassembly length (see [10] for more information). To
avoid these limitations, RTP/JPEG defines a simple fragmentation and
reassembly scheme at the RTP level.

3. RTP/JPEG Packet Format

   The RTP timestamp is in units of 90000Hz. The same timestamp MUST
   appear in each fragment of a given frame. The RTP marker bit MUST be
   set in the last packet of a frame.

3.1. JPEG header

   Each packet contains a special JPEG header which immediately follows
   the RTP header. The first 8 bytes of this header, called the "main
   JPEG header", are as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type-specific |              Fragment Offset                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |       Q       |     Width     |     Height    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
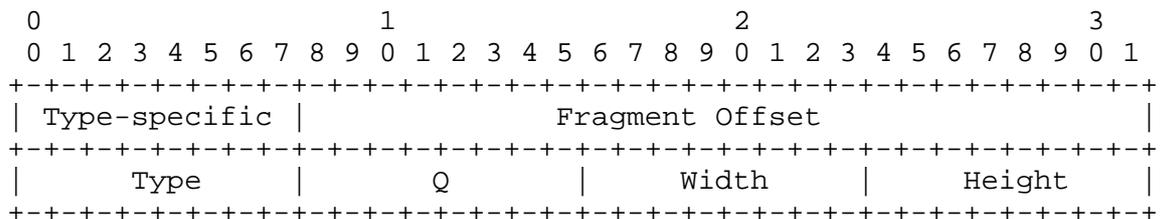
   All fields in this header except for the Fragment Offset field MUST
   remain the same in all packets that correspond to the same JPEG
   frame.

   A Restart Marker header and/or Quantization Table header may follow
   this header, depending on the values of the Type and Q fields.

3.1.1. Type-specific: 8 bits

   Interpretation depends on the value of the type field. If no
   interpretation is specified, this field MUST be zeroed on
   transmission and ignored on reception.

3.1.2. Fragment Offset: 24 bits

   The Fragment Offset is the offset in bytes of the current packet in
   the JPEG frame data. This value is encoded in network byte order
   (most significant byte first). The Fragment Offset plus the length of
   the payload data in the packet MUST NOT exceed 2^{24} bytes.

3.1.3. Type: 8 bits

   The type field specifies the information that would otherwise be
   present in a JPEG abbreviated table-specification as well as the
   additional JFIF-style parameters not defined by JPEG. Types 0-63 are
   reserved as fixed, well-known mappings to be defined by this document
   and future revisions of this document. Types 64-127 are the same as
   types 0-63, except that restart markers are present in the JPEG data
   and a Restart Marker header appears immediately following the main
   JPEG header. Types 128-255 are free to be dynamically defined by a
   session setup protocol (which is beyond the scope of this document).

3.1.4. Q: 8 bits

   The Q field defines the quantization tables for this frame. Q values
   0-127 indicate the quantization tables are computed using an
   algorithm determined by the Type field (see below). Q values 128-255
   indicate that a Quantization Table header appears after the main JPEG
   header (and the Restart Marker header, if present) in the first
   packet of the frame (fragment offset 0). This header can be used to
   explicitly specify the quantization tables in-band.

3.1.5. Width: 8 bits

   This field encodes the width of the image in 8-pixel multiples (e.g.,
   a width of 40 denotes an image 320 pixels wide). The maximum width
   is 2040 pixels.

3.1.6. Height: 8 bits

   This field encodes the height of the image in 8-pixel multiples
   (e.g., a height of 30 denotes an image 240 pixels tall). When
   encoding interlaced video, this is the height of a video field, since
   fields are individually JPEG encoded. The maximum height is 2040
   pixels.

3.1.7. Restart Marker header

   This header MUST be present immediately after the main JPEG header
   when using types 64-127. It provides the additional information
   required to properly decode a data stream containing restart markers.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Restart Interval        |F|L|       Restart Count       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
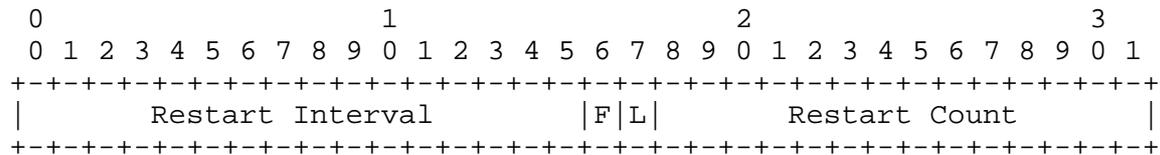
   The Restart Interval field specifies the number of MCUs that appear
   between restart markers. It is identical to the 16 bit value that
   would appear in the DRI marker segment of a JFIF header. This value
   MUST NOT be zero.

   If the restart intervals in a frame are not guaranteed to be aligned
   with packet boundaries, the F (first) and L (last) bits MUST be set
   to 1 and the Restart Count MUST be set to 0x3FFF. This indicates
   that a receiver MUST reassemble the entire frame before decoding it.

   To support partial frame decoding, the frame is broken into "chunks"
   each containing an integral number of restart intervals. The Restart
   Count field contains the position of the first restart interval in
   the current "chunk" so that receivers know which part of the frame
   this data corresponds to. A Restart Interval value SHOULD be chosen
   to allow a "chunk" to completely fit within a single packet. In this
   case, both the F and L bits of the packet are set to 1. However, if
   a chunk needs to be spread across multiple packets, the F bit will be
   set to 1 in the first packet of the chunk (and only that one) and the
   L bit will be set to 1 in the last packet of the chunk (and only that
   one).

3.1.8. Quantization Table header

   This header MUST be present after the main JPEG header (and after the
   Restart Marker header, if present) when using Q values 128-255. It
   provides a way to specify the quantization tables associated with
   this Q value in-band.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     MBZ       |   Precision   |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Quantization Table Data                    |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
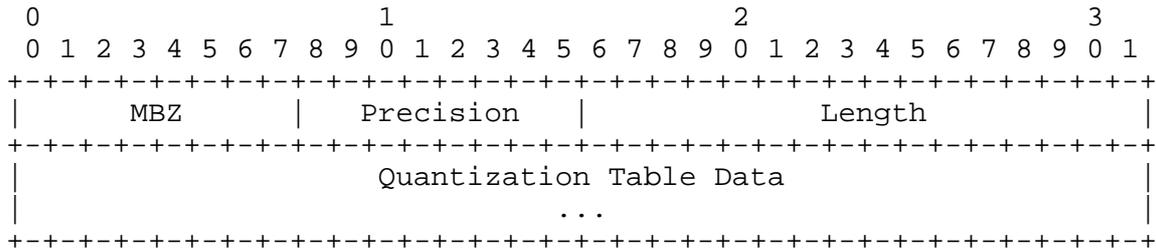
The Length field is set to the length in bytes of the quantization
table data to follow. The Length field MAY be set to zero to
indicate that no quantization table data is included in this frame.
See section 4.2 for more information. If the Length field in a
received packet is larger than the remaining number of bytes, the
packet MUST be discarded.

When table data is included, the number of tables present depends on
the JPEG type field. For example, type 0 uses two tables (one for
the luminance component and one shared by the chrominance
components). Each table is an array of 64 values given in zig-zag
order, identical to the format used in a JFIF DQT marker segment.

For each quantization table present, a bit in the Precision field
specifies the size of the coefficients in that table. If the bit is
zero, the coefficients are 8 bits yielding a table length of 64
bytes. If the bit is one, the coefficients are 16 bits for a table
length of 128 bytes. For 16 bit tables, the coefficients are
presented in network byte order. The rightmost bit in the Precision
field (bit 15 in the diagram above) corresponds to the first table
and each additional table uses the next bit to the left. Bits beyond
those corresponding to the tables needed by the type in use MUST be
ignored.

For Q values from 128 to 254, the Q value to quantization table data
mapping MUST be static, i.e., the receivers are guaranteed that they
only need to read the table data once in order to correctly decode
frames sent with that Q value. A Q value of 255 denotes that the
quantization table mapping is dynamic and can change on every frame.
Decoders MUST NOT depend on any previous version of the tables, and
need to reload these tables on every frame. Packets MUST NOT contain
Q = 255 and Length = 0.

3.1.9. JPEG Payload

The data following the RTP/JPEG headers is an entropy-coded segment
consisting of a single scan. The scan header is not present and is
inferred from the RTP/JPEG header. The scan is terminated either
implicitly (i.e., the point at which the image is fully parsed), or

explicitly with an EOI marker. The scan may be padded to arbitrary
length with undefined bytes. (Some existing hardware codecs generate
extra lines at the bottom of a video frame and removal of these lines
would require a Huffman-decoding pass over the data.)

The type code determines whether restart markers are present. If a
type supports restart markers, the packet MUST contain a non-zero
Restart Interval value in a Restart Marker Header and restart markers
MUST appear on byte aligned boundaries beginning with an 0xFF between
MCUs at that interval. Additional 0xFF bytes MAY appear between
restart intervals. This can be used in the packetization process to
align data to something like a word boundary for more efficient
copying. Restart markers MUST NOT appear anywhere else in the JPEG
payload. Types which do not support restart makers MUST NOT contain
restart markers anywhere in the JPEG payload. All packets MUST
contain a "stuffed" 0x00 byte following any true 0xFF byte generated
by the entropy coder [1, Sec. B.1.1.5].

4. Discussion

4.1. The Type Field

The Type field defines the abbreviated table-specification and
additional JFIF-style parameters not defined by JPEG, since they are
not present in the body of the transmitted JPEG data.

Three ranges of the type field are currently defined. Types 0-63 are
reserved as fixed, well-known mappings to be defined by this document
and future revisions of this document. Types 64-127 are the same as
types 0-63, except that restart markers are present in the JPEG data
and a Restart Marker header appears immediately following the main
JPEG header. Types 128-255 are free to be dynamically defined by a
session setup protocol (which is beyond the scope of this document).

Of the first group of fixed mappings, types 0 and 1 are currently
defined, along with the corresponding types 64 and 65 that indicate
the presence of restart markers. They correspond to an abbreviated
table-specification indicating the "Baseline DCT sequential" mode,
8-bit samples, square pixels, three components in the YUV color
space, standard Huffman tables as defined in [1, Annex K.3], and a
single interleaved scan with a scan component selector indicating
components 1, 2, and 3 in that order. The Y, U, and V color planes
correspond to component numbers 1, 2, and 3, respectively. Component
1 (i.e., the luminance plane) uses Huffman table number 0 and
quantization table number 0 (defined below) and components 2 and 3
(i.e., the chrominance planes) use Huffman table number 1 and
quantization table number 1 (defined below).

Type numbers 2-5 are reserved and SHOULD NOT be used. Applications
based on previous versions of this document (RFC 2035) should be
updated to indicate the presence of restart markers with type 64 or
65 and the Restart Marker header.

The two RTP/JPEG types currently defined are described below:

```
                        horizontal   vertical   Quantization
           types    component samp. fact. samp. fact. table number
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |       | 1 (Y) |     2      |    1     |      0      |
         | 0, 64 | 2 (U) |     1      |    1     |      1      |
         |       | 3 (V) |     1      |    1     |      1      |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |       | 1 (Y) |     2      |    2     |      0      |
         | 1, 65 | 2 (U) |     1      |    1     |      1      |
         |       | 3 (V) |     1      |    1     |      1      |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

These sampling factors indicate that the chrominance components of
type 0 video is downsampled horizontally by 2 (often called 4:2:2)
while the chrominance components of type 1 video are downsampled both
horizontally and vertically by 2 (often called 4:2:0).

Types 0 and 1 can be used to carry both progressively scanned and
interlaced image data. This is encoded using the Type-specific field
in the main JPEG header. The following values are defined:

   0 : Image is progressively scanned. On a computer monitor, it can
       be displayed as-is at the specified width and height.

   1 : Image is an odd field of an interlaced video signal. The
       height specified in the main JPEG header is half of the height
       of the entire displayed image. This field should be de-
       interlaced with the even field following it such that lines
       from each of the images alternate. Corresponding lines from
       the even field should appear just above those same lines from
       the odd field.

   2 : Image is an even field of an interlaced video signal.

   3 : Image is a single field from an interlaced video signal, but
       it should be displayed full frame as if it were received as
       both the odd & even fields of the frame. On a computer
       monitor, each line in the image should be displayed twice,
       doubling the height of the image.

Appendix B contains C source code for transforming the RTP/JPEG
header parameters into the JPEG frame and scan headers that are
absent from the data payload.

4.2. The Q Field

For JPEG types 0 and 1 (and their corresponding types 64 and 65), Q
values between 1 and 99 inclusive are defined as follows. Other
values less than 128 are reserved. Additional types are encouraged
to use this definition if applicable.

Both type 0 and type 1 JPEG require two quantization tables. These
tables are calculated as follows. For 1 <= Q <= 99, the Independent
JPEG Group's formula [5] is used to produce a scale factor S as:

        S = 5000 / Q   for 1 <= Q <= 50
          = 200 - 2 * Q   for 51 <= Q <= 99

This value is then used to scale Tables K.1 and K.2 from [1]
(saturating each value to 8 bits) to give quantization table numbers
0 and 1, respectively. C source code is provided in Appendix A to
compute these tables.

For Q values 128-255, dynamically defined quantization tables are
used. These tables may be specified either in-band or out of band by
something like a session setup protocol, but the Quantization Table
header MUST be present in the first packet of every frame. When the
tables are specified out of band, they may be omitted from the packet
by setting the Length field in this header to 0.

When the quantization tables are sent in-band, they need not be sent
with every frame. Like the out of band case, frames which do not
contain tables will have a Quantization Table header with a Length
field of 0. While this does decrease the overhead of including the
tables, new receivers will be unable to properly decode frames from
the time they start up until they receive the tables.

4.3. Fragmentation and Reassembly

Since JPEG frames can be large, they must often be fragmented.
Frames SHOULD be fragmented into packets in a manner avoiding
fragmentation at a lower level. If support for partial frame
decoding is desired, frames SHOULD be fragmented such that each
packet contains an integral number of restart intervals (see below).

Each packet that makes up a single frame MUST have the same
timestamp, and the RTP marker bit MUST be set on the last packet in a
frame. The fragment offset field of each packet is set to the byte

offset of its payload data within the original frame. Packets making
up a frame SHOULD be sent sequentially and the fragments they contain
MUST NOT overlap one another.

An entire frame can be identified as a sequence of packets beginning
with a packet having a zero fragment offset and ending with a packet
having the RTP marker bit set. Missing packets can be detected
either with RTP sequence numbers or with the fragment offset and
lengths of each packet. Reassembly could be carried out without the
offset field (i.e., using only the RTP marker bit and sequence
numbers), but an efficient single-copy implementation would not
otherwise be possible in the presence of misordered packets.
Moreover, if the last packet of the previous frame (containing the
marker bit) were dropped, then a receiver could not always detect
that the current frame is entirely intact.

4.4. Restart Markers

Restart markers indicate a point in the JPEG stream at which the
Huffman decoder and DC predictors are reset, allowing partial
decoding starting at that point. To fully take advantage of this,
however, a decoder must know which MCUs of a frame a particular
restart interval encodes. While the original JPEG specification does
provide a small sequence number field in the restart markers for this
purpose, it is not large enough to properly cope with the loss of an
entire packet's worth of data at a typical network MTU size. The
RTP/JPEG Restart Marker header contains the additional information
needed to accomplish this.

The size of restart intervals SHOULD be chosen to always allow an
integral number of restart intervals to fit within a single packet.
This will guarantee that packets can be decoded independently from
one another. If a restart interval ends up being larger than a
packet, the F and L bits in the Restart Marker header can be used to
fragment it, but the resulting set of packets must all be received by
a decoder for that restart interval to be decoded properly.

Once a decoder has received either a single packet with both the F
and L bits set on or a contiguous sequence of packets (based on the
RTP sequence number) which begin with an F bit and end with an L bit,
it can begin decoding. The position of the MCU at the beginning of
the data can be determined by multiplying the Restart Count value by
the Restart Interval value. A packet (or group of packets as
identified by the F and L bits) may contain any number of consecutive
restart intervals.

To accommodate encoders which generate frames with restart markers in
them but cannot fragment the data in this manner, the Restart Count

field may be set to 0x3FFF with the F and L bits both set to 1. This
indicates to decoders that the entire frame must be reassembled
before decoding it.

5. Security Considerations

RTP packets using the payload format defined in this specification
are subject to the security considerations discussed in the RTP
specification [6], and any appropriate RTP profile (for example [7]).
This implies that confidentiality of the media streams is achieved by
encryption. Because the data compression used with this payload
format is applied end-to-end, encryption may be performed after
compression so there is no conflict between the two operations.

A potential denial-of-service threat exists for data encodings using
compression techniques that have non-uniform receiver-end
computational load. The attacker can inject pathological datagrams
into the stream which are complex to decode and cause the receiver to
be overloaded. However, this encoding does not exhibit any
significant non-uniformity.

Another potential denial-of-service threat exists around the
fragmentation mechanism presented here. Receivers should be prepared
to limit the total amount of data associated with assembling received
frames so as to avoid resource exhaustion.

As with any IP-based protocol, in some circumstances a receiver may
be overloaded simply by the receipt of too many packets, either
desired or undesired. Network-layer authentication may be used to
discard packets from undesired sources, but the processing cost of
the authentication itself may be too high. In a multicast
environment, pruning of specific sources will be implemented in a
future version of IGMP [8] and in multicast routing protocols to
allow a receiver to select which sources are allowed to reach it.

A security review of this payload format found no additional
considerations beyond those in the RTP specification.

6. Authors' Addresses

    Lance M. Berc
    Systems Research Center
    Digital Equipment Corporation
    130 Lytton Ave
    Palo Alto CA 94301

    Phone: +1 650 853 2100
    EMail: berc@pa.dec.com

    William C. Fenner
    Xerox PARC
    3333 Coyote Hill Road
    Palo Alto, CA 94304

    Phone: +1 650 812 4816
    EMail: fenner@parc.xerox.com

    Ron Frederick
    Xerox PARC
    3333 Coyote Hill Road
    Palo Alto, CA 94304

    Phone: +1 650 812 4459
    EMail: frederick@parc.xerox.com

    Steven McCanne
    University of California at Berkeley
    Electrical Engineering and Computer Science
    633 Soda Hall
    Berkeley, CA 94720

    Phone: +1 510 642 0865
    EMail: mccanne@cs.berkeley.edu

    Paul Stewart
    Xerox PARC
    3333 Coyote Hill Road
    Palo Alto, CA 94304

    Phone: +1 650 812 4821
    EMail: stewart@parc.xerox.com

7. References

   [1]   ISO DIS 10918-1. Digital Compression and Coding of Continuous-
         tone Still Images (JPEG), CCITT Recommendation T.81.

   [2]   William B. Pennebaker, Joan L. Mitchell, JPEG: Still Image Data
         Compression Standard, Van Nostrand Reinhold, 1993.

   [3]   Gregory K. Wallace, The JPEG Sill Picture Compression Standard,
         Communications of the ACM, April 1991, Vol 34, No. 1, pp. 31-44.

   [4]   The JPEG File Interchange Format. Maintained by C-Cube
         Microsystems, Inc., and available in
         ftp://ftp.uu.net/graphics/jpeg/jfif.ps.gz.

   [5]   Tom Lane et. al., The Independent JPEG Group software JPEG
         codec. Source code available in
         ftp://ftp.uu.net/graphics/jpeg/jpegsrc.v6a.tar.gz.

   [6]   Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson,
         "RTP: A Transport Protocol for Real-Time Applications", RFC
         1889, January 1996.

   [7]   Schulzrinne, H., "RTP Profile for Audio and Video Conferences
         with Minimal Control", RFC 1890, January 1996.

   [8]   Fenner, W., "Internet Group Management Protocol Version 2", RFC
         2236, November 1997.

   [9]   Bradner, S., "Key words for use in RFCs to Indicate Requirement
         Levels", BCP 14, RFC 2119, March 1997.

   [10] Kent C., and J. Mogul, "Fragmentation Considered Harmful",
         Proceedings of the ACM SIGCOMM '87 Workshop on Frontiers in
         Computer Communications Technology, August 1987.

Appendix A

   The following code can be used to create a quantization table from a
   Q factor:

```
/*
 * Table K.1 from JPEG spec.
 */
static const int jpeg_luma_quantizer[64] = {
        16, 11, 10, 16, 24, 40, 51, 61,
        12, 12, 14, 19, 26, 58, 60, 55,
        14, 13, 16, 24, 40, 57, 69, 56,
        14, 17, 22, 29, 51, 87, 80, 62,
        18, 22, 37, 56, 68, 109, 103, 77,
        24, 35, 55, 64, 81, 104, 113, 92,
        49, 64, 78, 87, 103, 121, 120, 101,
        72, 92, 95, 98, 112, 100, 103, 99
};

/*
 * Table K.2 from JPEG spec.
 */
static const int jpeg_chroma_quantizer[64] = {
        17, 18, 24, 47, 99, 99, 99, 99,
        18, 21, 26, 66, 99, 99, 99, 99,
        24, 26, 56, 99, 99, 99, 99, 99,
        47, 66, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99,
        99, 99, 99, 99, 99, 99, 99, 99
};

/*
 * Call MakeTables with the Q factor and two u_char[64] return arrays
 */
void
MakeTables(int q, u_char *lqt, u_char *cqt)
{
  int i;
  int factor = q;

  if (q < 1) factor = 1;
  if (q > 99) factor = 99;
  if (q < 50)
    q = 5000 / factor;
  else
    q = 200 - factor*2;
```

```
  for (i=0; i < 64; i++) {
     int lq = (jpeg_luma_quantizer[i] * q + 50) / 100;
     int cq = (jpeg_chroma_quantizer[i] * q + 50) / 100;

     /* Limit the quantizers to 1 <= q <= 255 */
     if (lq < 1) lq = 1;
     else if (lq > 255) lq = 255;
     lqt[i] = lq;

     if (cq < 1) cq = 1;
     else if (cq > 255) cq = 255;
     cqt[i] = cq;
  }
}
```

Appendix B

   The following routines can be used to create the JPEG marker segments
   corresponding to the table-specification data that is absent from the
   RTP/JPEG body.

```
u_char lum_dc_codelens[] = {
        0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
};

u_char lum_dc_symbols[] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
};

u_char lum_ac_codelens[] = {
        0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d,
};

u_char lum_ac_symbols[] = {
        0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
        0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
        0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
        0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
        0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
        0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
        0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
        0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
        0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
        0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
        0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
        0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
        0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
        0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
        0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
        0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
        0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
        0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
        0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
        0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
        0xf9, 0xfa,
};

u_char chm_dc_codelens[] = {
        0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
};

u_char chm_dc_symbols[] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
```

```
};

u_char chm_ac_codelens[] = {
        0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77,
};

u_char chm_ac_symbols[] = {
        0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
        0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
        0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
        0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 0xf0,
        0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,
        0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,
        0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,
        0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
        0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
        0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
        0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
        0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
        0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,
        0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,
        0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,
        0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,
        0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,
        0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,
        0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
        0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
        0xf9, 0xfa,
};

u_char *
MakeQuantHeader(u_char *p, u_char *qt, int tableNo)
{
        *p++ = 0xff;
        *p++ = 0xdb;               /* DQT */
        *p++ = 0;                  /* length msb */
        *p++ = 67;                 /* length lsb */
        *p++ = tableNo;
        memcpy(p, qt, 64);
        return (p + 64);
}

u_char *
MakeHuffmanHeader(u_char *p, u_char *codelens, int ncodes,
                  u_char *symbols, int nsymbols, int tableNo,
                  int tableClass)
{
        *p++ = 0xff;
```

```
        *p++ = 0xc4;                 /* DHT */
        *p++ = 0;                    /* length msb */
        *p++ = 3 + ncodes + nsymbols; /* length lsb */
        *p++ = (tableClass << 4) | tableNo;
        memcpy(p, codelens, ncodes);
        p += ncodes;
        memcpy(p, symbols, nsymbols);
        p += nsymbols;
        return (p);
}

u_char *
MakeDRIHeader(u_char *p, u_short dri) {
        *p++ = 0xff;
        *p++ = 0xdd;                 /* DRI */
        *p++ = 0x0;                  /* length msb */
        *p++ = 4;                    /* length lsb */
        *p++ = dri >> 8;             /* dri msb */
        *p++ = dri & 0xff;           /* dri lsb */
        return (p);
}

/*
 *  Arguments:
 *     type, width, height: as supplied in RTP/JPEG header
 *     lqt, cqt: quantization tables as either derived from
 *         the Q field using MakeTables() or as specified
 *         in section 4.2.
 *     dri: restart interval in MCUs, or 0 if no restarts.
 *
 *     p: pointer to return area
 *
 *  Return value:
 *     The length of the generated headers.
 *
 *     Generate a frame and scan headers that can be prepended to the
 *     RTP/JPEG data payload to produce a JPEG compressed image in
 *     interchange format (except for possible trailing garbage and
 *     absence of an EOI marker to terminate the scan).
 */
int MakeHeaders(u_char *p, int type, int w, int h, u_char *lqt,
                u_char *cqt, u_short dri)
{
        u_char *start = p;

        /* convert from blocks to pixels */
        w <<= 3;
        h <<= 3;
```

```
        *p++ = 0xff;
        *p++ = 0xd8;                 /* SOI */

        p = MakeQuantHeader(p, lqt, 0);
        p = MakeQuantHeader(p, cqt, 1);

        if (dri != 0)
                p = MakeDRIHeader(p, dri);

        *p++ = 0xff;
        *p++ = 0xc0;                 /* SOF */
        *p++ = 0;                    /* length msb */
        *p++ = 17;                   /* length lsb */
        *p++ = 8;                    /* 8-bit precision */
        *p++ = h >> 8;               /* height msb */
        *p++ = h;                    /* height lsb */
        *p++ = w >> 8;               /* width msb */
        *p++ = w;                    /* wudth lsb */
        *p++ = 3;                    /* number of components */
        *p++ = 0;                    /* comp 0 */
        if (type == 0)
                *p++ = 0x21;     /* hsamp = 2, vsamp = 1 */
        else
                *p++ = 0x22;     /* hsamp = 2, vsamp = 2 */
        *p++ = 0;                    /* quant table 0 */
        *p++ = 1;                    /* comp 1 */
        *p++ = 0x11;                 /* hsamp = 1, vsamp = 1 */
        *p++ = 1;                    /* quant table 1 */
        *p++ = 2;                    /* comp 2 */
        *p++ = 0x11;                 /* hsamp = 1, vsamp = 1 */
        *p++ = 1;                    /* quant table 1 */
        p = MakeHuffmanHeader(p, lum_dc_codelens,
                              sizeof(lum_dc_codelens),
                              lum_dc_symbols,
                              sizeof(lum_dc_symbols), 0, 0);
        p = MakeHuffmanHeader(p, lum_ac_codelens,
                              sizeof(lum_ac_codelens),
                              lum_ac_symbols,
                              sizeof(lum_ac_symbols), 0, 1);
        p = MakeHuffmanHeader(p, chm_dc_codelens,
                              sizeof(chm_dc_codelens),
                              chm_dc_symbols,
                              sizeof(chm_dc_symbols), 1, 0);
        p = MakeHuffmanHeader(p, chm_ac_codelens,
                              sizeof(chm_ac_codelens),
                              chm_ac_symbols,
                              sizeof(chm_ac_symbols), 1, 1);
```

```
        *p++ = 0xff;
        *p++ = 0xda;                    /* SOS */
        *p++ = 0;                       /* length msb */
        *p++ = 12;                      /* length lsb */
        *p++ = 3;                       /* 3 components */
        *p++ = 0;                       /* comp 0 */
        *p++ = 0;                       /* huffman table 0 */
        *p++ = 1;                       /* comp 1 */
        *p++ = 0x11;                    /* huffman table 1 */
        *p++ = 2;                       /* comp 2 */
        *p++ = 0x11;                    /* huffman table 1 */
        *p++ = 0;                       /* first DCT coeff */
        *p++ = 63;                      /* last DCT coeff */
        *p++ = 0;                       /* sucessive approx. */

        return (p - start);
};
```

Appendix C

   The following routine is used to illustrate the RTP/JPEG packet
   fragmentation and header creation.

   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 data header from RFC1889
 */
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;                /* timestamp */
        u_int32 ssrc;              /* synchronization source */
        u_int32 csrc[1];           /* optional CSRC list */
} rtp_hdr_t;

#define RTP_HDR_SZ 12

/* The following definition is from RFC1890 */
#define RTP_PT_JPEG             26

struct jpeghdr {
        unsigned int tspec:8;    /* type-specific field */
        unsigned int off:24;     /* fragment byte offset */
        u_int8 type;             /* id of jpeg decoder params */
        u_int8 q;                /* quantization factor (or table id) */
        u_int8 width;            /* frame width in 8 pixel blocks */
        u_int8 height;           /* frame height in 8 pixel blocks */
};

struct jpeghdr_rst {
        u_int16 dri;
        unsigned int f:1;
        unsigned int l:1;
        unsigned int count:14;
};
```

```
struct jpeghdr_qtable {
        u_int8  mbz;
        u_int8  precision;
        u_int16 length;
};

#define RTP_JPEG_RESTART           0x40

/* Procedure SendFrame:
 *
 *   Arguments:
 *     start_seq: The sequence number for the first packet of the current
 *                frame.
 *     ts: RTP timestamp for the current frame
 *     ssrc: RTP SSRC value
 *     jpeg_data: Huffman encoded JPEG scan data
 *     len: Length of the JPEG scan data
 *     type: The value the RTP/JPEG type field should be set to
 *     typespec: The value the RTP/JPEG type-specific field should be set
 *                to
 *     width: The width in pixels of the JPEG image
 *     height: The height in pixels of the JPEG image
 *     dri: The number of MCUs between restart markers (or 0 if there
 *          are no restart markers in the data
 *     q: The Q factor of the data, to be specified using the Independent
 *        JPEG group's algorithm if 1 <= q <= 99, specified explicitly
 *        with lqt and cqt if q >= 128, or undefined otherwise.
 *     lqt: The quantization table for the luminance channel if q >= 128
 *     cqt: The quantization table for the chrominance channels if
 *          q >= 128
 *
 *   Return value:
 *     the sequence number to be sent for the first packet of the next
 *     frame.
 *
 * The following are assumed to be defined:
 *
 * PACKET_SIZE                            - The size of the outgoing packet
 * send_packet(u_int8 *data, int len)  - Sends the packet to the network
 */

u_int16 SendFrame(u_int16 start_seq, u_int32 ts, u_int32 ssrc,
                  u_int8 *jpeg_data, int len, u_int8 type,
                  u_int8 typespec, int width, int height, int dri,
                  u_int8 q, u_int8 *lqt, u_int8 *cqt) {
        rtp_hdr_t rtphdr;
        struct jpeghdr jpghdr;
        struct jpeghdr_rst rsthdr;
```

```
            struct jpeghdr_qtable qtblhdr;
            u_int8 packet_buf[PACKET_SIZE];
            u_int8 *ptr;
            int bytes_left = len;
            int seq = start_seq;
            int pkt_len, data_len;

            /* Initialize RTP header
             */
            rtphdr.version = 2;
            rtphdr.p = 0;
            rtphdr.x = 0;
            rtphdr.cc = 0;
            rtphdr.m = 0;
            rtphdr.pt = RTP_PT_JPEG;
            rtphdr.seq = start_seq;
            rtphdr.ts = ts;
            rtphdr.ssrc = ssrc;

            /* Initialize JPEG header
             */
            jpghdr.tspec = typespec;
            jpghdr.off = 0;
            jpghdr.type = type | ((dri != 0) ? RTP_JPEG_RESTART : 0);
            jpghdr.q = q;
            jpghdr.width = width / 8;
            jpghdr.height = height / 8;

            /* Initialize DRI header
             */
            if (dri != 0) {
                    rsthdr.dri = dri;
                    rsthdr.f = 1;          /* This code does not align RIs */
                    rsthdr.l = 1;
                    rsthdr.count = 0x3fff;
            }

            /* Initialize quantization table header
             */
            if (q >= 128) {
                    qtblhdr.mbz = 0;
                    qtblhdr.precision = 0; /* This code uses 8 bit tables only *
/
                    qtblhdr.length = 128;  /* 2 64-byte tables */
            }

            while (bytes_left > 0) {
                    ptr = packet_buf + RTP_HDR_SZ;
                    memcpy(ptr, &jpghdr, sizeof(jpghdr));
```

```
                ptr += sizeof(jpghdr);

                if (dri != 0) {
                        memcpy(ptr, &rsthdr, sizeof(rsthdr));
                        ptr += sizeof(rsthdr);
                }

                if (q >= 128 && jpghdr.off == 0) {
                        memcpy(ptr, &qtblhdr, sizeof(qtblhdr));
                        ptr += sizeof(qtblhdr);
                        memcpy(ptr, lqt, 64);
                        ptr += 64;
                        memcpy(ptr, cqt, 64);
                        ptr += 64;
                }

                data_len = PACKET_SIZE - (ptr - packet_buf);
                if (data_len >= bytes_left) {
                        data_len = bytes_left;
                        rtphdr.m = 1;
                }

                memcpy(packet_buf, &rtphdr, RTP_HDR_SZ);
                memcpy(ptr, jpeg_data + jpghdr.off, data_len);

                send_packet(packet_buf, (ptr - packet_buf) + data_len);

                jpghdr.off += data_len;
                bytes_left -= data_len;
                rtphdr.seq++;
        }
        return rtphdr.seq;
}
```

Appendix D

   This section outlines the changes between this document and its
   precdecessor, RFC 2035. The changes to the protocol were made with
   an eye towards causing as few interoperability problems between
   implementations based on the older text and newer implementations,
   and indeed, many of the obsolete conventions can still be
   unambiguously decoded by a newer implementation. However, use of the
   older conventions in newer implementations is strongly discouraged.

   o   Types 0 and 1 have been augmented to allow for the encoding of
        interlaced video images, using 2 bits of the type-specific
        field. See section 4.1 for details.

   o   There has been discussion in the working group arguing for more
        flexibility in specifying the JPEG quantization tables. This
        memo allows table coefficients to be specified explicitly
        through the use of an optional Quantization Table header,
        discussed in sections 3.1.8 and 4.2.

   o   In RFC 2035, the encoding of restart marker information in the
        Type field made it difficult to add new types. Additionally, the
        type- specific field was used for the restart count, making it
        unavailable for other type-specific purposes. This memo moves
        the restart marker indication to a particular bit in the Type
        field, and adds an optional header to hold the additional
        information required, leaving the type-specific field free for
        its intended purpose. The handling of partial frame decoding
        was also made more robust against packet loss. See sections
        3.1.7 and 4.4 for details.

Full Copyright Statement