Network Working Group
Request for Comments: 2733
Category: Standards Track

J. Rosenberg
dynamicsoft
H. Schulzrinne
Columbia University

December 1999

An RTP Payload Format for Generic Forward Error Correction

Status of this Memo

Copyright Notice

Abstract

This document specifies a payload format for generic forward error correction of media encapsulated in RTP. It is engineered for FEC algorithms based on the exclusive-or (parity) operation. The payload format allows end systems to transmit using arbitrary block lengths and parity schemes. It also allows for the recovery of both the payload and critical RTP header fields. Since FEC is sent as a separate stream, it is backwards compatible with non-FEC capable hosts, so that receivers which do not wish to implement FEC can just ignore the extensions.

Table of Contents

1 Introduction

The quality of packet voice on the Internet has been mediocre due, in part, to high packet loss rates. This is especially true on wide-area connections. Unfortunately, the strict delay requirements of real-time multimedia usually eliminate the possibility of retransmissions.

It is for this reason that forward error correction (FEC) has been proposed to compensate for packet loss in the Internet [1] [2]. In particular, the use of traditional error correcting codes, such as parity, Reed-Solomon, and Hamming codes, has attracted attention. To support these mechanisms, protocol support is required.

This document defines a payload format for RTP [3] which allows for generic forward error correction of real time media. In this context, generic means that the FEC protocol is (1) independent of the nature of the media being protected, be it audio, video, or otherwise, (2) flexible enough to support a wide variety of FEC mechanisms, (3) designed for adaptivity so that the FEC technique can be modified easily without out of band signaling, and (4) supportive of a number of different mechanisms for transporting the FEC packets.

2 Terminology

The following terms are used throughout this document:

Media Payload: is a piece of raw, un-protected user data which is to be transmitted from the sender. The media payload is placed inside of an RTP packet.

Media Header: is the RTP header for the packet containing the media payload.

Media Packet: The combination of a media payload and media header is called a media packet.

FEC Packet: The forward error correction algorithms at the transmitter take the media packets as an input. They output both the media packets that they are passed, and new packets called FEC packets. The FEC packets are formatted according to the rules specified in this document.

FEC Header: The FEC header is the header information contained in an FEC packet.

FEC Payload: The FEC payload is the payload in an FEC packet.

Associated: An FEC packet is said to be "associated" with one or more media packets when those media packets are used to generate the FEC packet (by use of the exclusive or operation).

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 [4].

3 Basic Operation

The payload format described here is used whenever a participant in an RTP session would like to protect a media stream it is sending with forward error correction (FEC). The FEC supported by the format are those codes based on simple exclusive or (xor) parities. The sender takes some set of packets from the media stream, and applies an xor operation across the payloads. The sender also applies the xor operation over components of the RTP headers. Based on the procedures defined here, the result is an RTP packet containing FEC information. This packet can be used at the receiver to recover any one of the packets used to generate the FEC packet. This document does not mandate the particular set of media packets combined to generate an FEC packet (such a set [is] referred to as a code). Use of differing sets results in a tradeoff between overhead, delay, and recoverability. Section 4 outlines some possible combinations.

The payload format contains information that allows the sender to tell the receiver exactly which media packets have been used to generate the FEC. Specifically, each FEC packet contains a bitmask, called the offset mask, containing 24 bits. If bit i in the mask is set to 1, the media packet with sequence number N + i was used to generate this FEC packet. N is called the sequence number base, and is sent in the FEC packet as well. The offset mask and payload type are sufficient to signal arbitrary parity based forward error correction schemes with little overhead.

This document also describes procedures that allow the receiver to make use of the FEC without having to know the details of specific codes. This allows the sender much flexibility; it can adapt the code in use based on network conditions, and be certain the receivers can still make use of the FEC for recovery.

As the sender generates FEC packets, they are sent to the receivers. The sender still usually sends the original media stream, as if there were no FEC. This allows the media stream to still be used by receivers who are not FEC capable. However, some FEC codes do not require the original media to be sent; the FEC stream is sufficient for recovery. These codes have the drawback that all receivers must be FEC capable. However, they are supported by this format.

The FEC packets are not sent in the same RTP stream as the media packets. They can be sent as a separate stream, or as a secondary codec in the redundant codec payload format [5]. When sent as a separate stream, the FEC packets have their own sequence number space. Although the timestamps for the FEC packets are derived from the media packets, they increment monotonically. FEC packet streams thus work well with any header compression mechanism which requires fixed deltas between fields in the packet header.

This document does not prescribe the definition of "separate streams", but leaves this to applications and higher level protocols to define. For multicast, the separate stream may be implemented by separate multicast groups, different ports in the same group, or by a different SSRC within the same group/port. For unicast, different ports or different SSRC may be used. Each of these approaches has drawbacks and benefits which depend on the application.

At the receiver, the FEC and original media are received. If no media packets are lost, the FEC can be ignored. In the event of loss, the FEC packets can be combined with other media and FEC packets that have been received, resulting in recovery of missing media packets. The recovery is exact; the payload is perfectly reconstructed, along with most components of the header.

RTP packets which contain data formatted according to this specification (i.e., FEC packets) are signaled using dynamic RTP payload types.

4 Parity Codes

For brevity, we define the function f(x,y,..) to be the XOR (parity) operator applied to the packets x,y,... The output of this function is another packet, called the parity packet. For simplicity, we assume here that the parity packet is computed as the bitwise XOR of the input packets. The exact procedure is specified in section 6.

Recovery of data packets using parity codes is accomplished by generating one or more parity packets over a group of data packets. To be effective, the parity packets must be generated by linearly independent combinations of data packets. The particular combination is called a parity code. One class of codes takes a group of k data packets, and generates n-k parity packets. There are a large number of possible parity codes for a given n,k. The payload format does not mandate a particular code.

For example, consider a parity code which generates a single parity packet over two data packets. If the original media packets are a,b,c,d, the packets generated by the sender are:

```
a        b        c        d              <-- media stream
           f(a,b)            f(c,d)       <-- FEC stream
```

where time increases to the right. In this example, the error correction scheme (we use the terms scheme and code interchangeably) introduces a 50% overhead. But if b is lost, a and f(a,b) can be used to recover b.

Some additional codes are listed below. In each, the original media stream consists of packets a,b,c,d and so on.

Scheme 1

This scheme is the similar to the one in the example above. However, instead of sending b, followed by f(a,b), f(a,b) is sent before b. Doing this clearly requires additional delay at the sender. However, if allows some bursts of two consecutive packet losses to be recovered. The packets generated by the sender look like:

```
a        b        c        d        e        <-- media stream
  f(a,b)   f(b,c)   f(c,d)   f(d,e)          <-- FEC stream
```

Scheme 2

It is not strictly necessary for the original media stream to be transmitted. In this scheme, only FEC packets are transmitted. This scheme allows for recovery of all single packet losses and some consecutive packet losses, but with slightly less overhead than scheme 1. The packets generated by the sender look like:

```
f(a,b)  f(a,c)  f(a,b,c)  f(c,d)  f(c,e)  f(c,d,e)  <-- FEC stream
```

Scheme 3

This scheme requires the receiver to wait an additional four packet intervals to recover the original media packets. However, it can recover from one, two or three consecutive packet losses. The packets generated by the sender look like:

```
a         b          c                    d     <-- media stream
            f(a,b,c)    f(a,c,d) f(a,b,d)       <-- FEC stream
```

5 RTP Media Packet Structure

The formatting of the media packets is unaffected by FEC. If the FEC is sent as a separate stream, the media packets are sent as if there was no FEC. If the FEC is being sent as a redundant codec, the media packets are sent as the main codec as defined in RFC 2198 [5].

This lends to a very efficient encoding. When little (or no) FEC is used, there are mostly media packets being sent. This means that the overhead (present in FEC packets only) tracks the amount of FEC in use.

6 FEC Packet Structure

An FEC packet is constructed by placing an FEC header and FEC payload in the RTP payload, as shown in Figure 1:

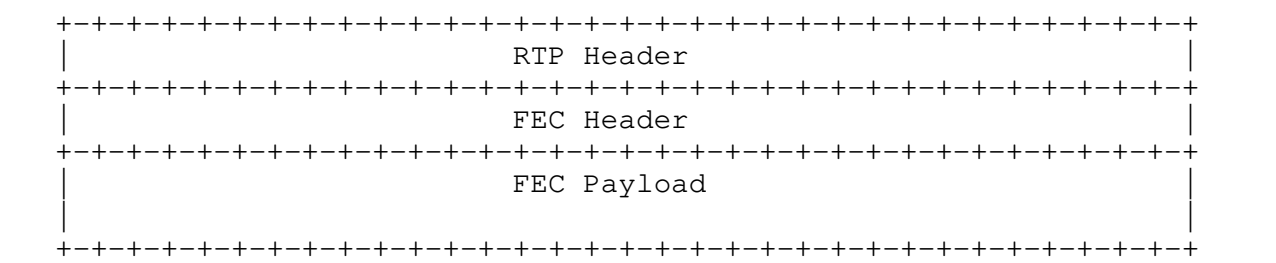


Figure 1: FEC Packet Structure

6.1 RTP Header of FEC Packets

The version field is set to 2. The padding bit is computed via the protection operation, defined below. The extension bit is also computed via the protection operation. The SSRC value will generally be the same as the SSRC value of the media stream it protects. It MAY be different if the FEC stream is being demultiplexed via the SSRC value. The CC value is computed via the protection operation. The CSRC list is never present, independent of the value of the CC field. The extension is never present, independent of the value of the X bit. The marker bit is computed via the protection operation.

The sequence number has the standard definition: it MUST be one higher than the sequence number in the previously transmitted FEC packet. The timestamp MUST be set to the value of the media RTP clock at the instant the FEC packet is transmitted. This results in the TS value in FEC packets to be monotonically increasing, independent of the FEC scheme.

The payload type for the FEC packet is determined through dynamic, out of band means. According to RFC 1889 [3], RTP participants which cannot recognize a payload type must discard it. This provides backwards compatibility. The FEC mechanisms can then be used in a multicast group with mixed FEC-capable and FEC-incapable receivers.

6.2 FEC Header

This header is 12 bytes. The format of the header is shown in Figure 2, and consists of an SN base field, length recovery field, E field, PT recovery field, mask field and TS recovery field.

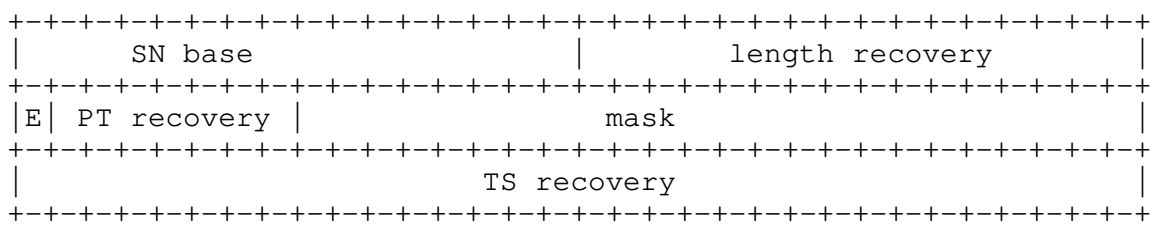


Figure 2: Parity Header Format

The length recovery field is used to determine the length of any recovered packets. It is computed via the protection operation applied to the unsigned network-ordered 16 bit representation of the sums of the lengths (in bytes) of the media payload, CSRC list, extension and padding of media packets associated with this FEC packet (in other words, the CSRC list, extension, and padding, if present, are "counted" as part of the payload). This allows the FEC procedure to be applied even when the lengths of the media packets are not identical. For example, assume an FEC packet is being generated by xor'ing two media packets together. The length of the two media packets are 3 (0b011) and 5 (0b101) bytes, respectively. The length recovery field is then encoded as 0b011 xor 0b101 = 0b110.

The E bit indicates a header extension. Implementations conforming to this version of the specification MUST set this bit to zero.

The PT recovery field is obtained via the protection operation applied to the payload type values of the media packets associated with the FEC packet.

The mask field is 24 bits. If bit i in the mask is set to 1, then the media packet with sequence number N + i is associated with this FEC packet, where N is the SN Base field in the FEC packet header. The least significant bit corresponds to i=0, and the most significant to i=23.

The SN base field MUST be set to the minimum sequence number of those media packets protected by FEC. This allows for the FEC operation to extend over any string of at most 24 packets.

The TS recovery field is computed via the protection operation applied to the timestamps of the media packets associated with this FEC packet. This allows the timestamp to be completely recovered.

The payload of the FEC packet is the protection operation applied to the concatenation of the CSRC list, RTP extension, media payload, and padding of the media packets associated with the FEC packet.

Note that it's possible for the FEC packet to be slightly larger than the media packets it protects (due to the presence of the FEC header). This could cause difficulties if this results in the FEC packet exceeding the Maximum Transmission Unit size for the path along which it is sent.

7 Protection Operation

The protection operation involves concatenating specific fields from the RTP header of the media packet, appending the payload, padding with zeroes, and then computing the xor across the resulting bit strings. The resulting bit string is used to generate the FEC packet.

The following procedure MAY be followed for the protection operation. Other procedures MAY be followed, but the end result MUST be identical to the one described here. For each media packet to be protected, a bit string is generated by concatenating the following fields together in the order specifed:

- Padding Bit (1 bit)
- Extension Bit (1 bit)
- CC bits (4 bits)
- Marker bit (1 bit)
- Payload Type (7 bits)
- Timestamp (32 bits)
- Unsigned network-ordered 16 bit representation of the sum of the lengths (in bytes) of the CSRC List, length of the padding, length of the extension, and length of the media payload (16 bits)
- if CC is nonzero, the CSRC List (variable length)
- if X is 1, the Header Extension (variable length)
- the payload (variable length)
- Padding, if present (variable length)

Note that the Padding Bit (first entry above) forms the most significant bit of the bit string.

If the lengths of the bit strings are not equal, each bit string that is shorter than the length of the longest, MUST be padded to the length of the longest. Any value for the pad may be used. The pad MUST be added at the end of the bit string.

The parity operation is then applied across the bit strings. The result is the bit string used to build the FEC packet. Call this the FEC bit string.

The first (most significant) bit in the FEC bit string is written into the Padding Bit of the FEC packet. The second bit in the FEC bit string is written into the Extension bit of the FEC packet. The next four bits of the FEC bit string are written into the CC field of the FEC packet. The next bit of the FEC bit string is written into the marker bit of the FEC packet. The next 7 bits of the FEC bit string are written into the PT recovery field in the FEC packet header. The next 32 bits of the FEC bit string are written into the TS recovery field in the packet header. The next 16 bits are written into the length recovery field in the FEC packet header. The remaining bits are set to be the payload of the FEC packet.

8 Recovery Procedures

The FEC packets allow end systems to recover from the loss of media packets. All of the header fields of the missing packets, including CSRC lists, extensions, padding bits, marker and payload type, are recoverable. This section describes the procedure for performing this recovery.

Recovery requires two distinct operations. The first determines which packets (media and FEC) must be combined in order to recover a missing packet. Once this is done, the second step is to actually reconstruct the data. The second step MUST be performed as described below. The first step MAY be based on any algorithm chosen by the implementer. Different algorithms result in a tradeoff between complexity and the ability to recover missing packets if at all possible.

8.1 Reconstruction

Let T be the list of packets (FEC and media) which can be combined to recover some media packet xi. The procedure is as follows:

1. For the media packets in T, compute the bit string as described in the protection operation of the previous section.

2. For the FEC packet in T, compute the bit string in the same fashion, except use the PT Recovery instead of Payload Type, TS Recovery instead of Timestamp, and always set the CSRC list, extension, and padding to null.

3. If any of the bit strings generated from the media packets are shorter than the bit string generated from the FEC packet, pad them to be the same length as the bit string generated from the FEC. The padding MUST be added at the end of the bit string, and MAY be of any value.

4. Perform the exclusive or (parity) operation across the bit strings, resulting in a recovery bit string.

5. Create a new packet with the standard 12 byte RTP header and no payload.

6. Set the version of the new packet to 2.

7. Set the Padding bit in the new packet to the first bit in the recovery bit string.

8. Set the Extension bit in the new packet to the second bit in the recovery bit string.

9. Set the CC field to the next four bits in the recovery bit string.

10. Set the marker bit in the new packet to the next bit in the recovery bit string.

11. Set the payload type in the new packet to the next 7 bits in the recovery bit string.

12. Set the SN field in the new packet to xi.

13. Set the TS field in the new packet to the next 32 bits in the recovery bit string.

14. Take the next 16 bits of the recovery bit string. Whatever unsigned integer this represents (assuming network-order), take that many bytes from the recovery bit string and append them to the new packet. This represents the CSRC list, extension, payload, and padding.

15. Set the SSRC of the new packet to the SSRC of the media stream it's protecting.

This procedure will completely recover both the header and payload of an RTP packet.

8.2 Determination of When to Recover

The previous section discussed how to recover a media packet with sequence number xi when all of the packets needed to recover it were available. The decision about whether to attempt recovery of some media packet xi, and how to determine if sufficient data is available to recover it, is left to the implementer. However, this section provides a simple algorithm which MAY be used for this purpose.

The algorithm is described below in C code. The code assumes that several functions exist. recover_packet() takes the sequence number of a packet, and an FEC packet. Using the FEC packet and data packets received previously, the data packet with the given sequence number is recovered. add_fec_to_pending_list() adds the given FEC packet to a linked list of FEC packets which have not yet been used for recovery. wait_for_packet() waits for a packet, FEC or data, from the network. remove_from_pending_list() removes the FEC packet from the pending list. The structure packet contains a boolean variable fec which is true when the packet is FEC, false if it's media. When its an FEC packet, the mask and snbase field contain those values from the FEC packet header. When it's a media packet, the sn variable contains the sequence number of the packet. The global array A indicates which media packets have been received, and which have not. It is indexed by the sequence number of the packet.

The function fec_recovery implements the algorithm. It waits for packets, and when it receives an FEC packet, calls recover_with_fec() to attempt to use it to recover. If no recovery is possible, the FEC packet is stored for later attempts. If the received packet was a media packet, its presence is noted, and any old FEC packets are checked to see if recovery is now possible. Recovered packets are treated as if they were received, triggering further attempts at recovery.

A real implementation will need to use a circular buffer instead of the simple array (A in the code) in order to avoid running off the end of the buffer. In addition, the code below does not attempt to free up FEC packets that are old and were never used. Normally, such discarding is done based on time constraints introduced by the playout buffer. If an FEC data protects packets whose play time has elapsed, the FEC is no longer needed.

```
typedef struct packet_s {

  BOOLEAN fec;               /* FEC or media */

  int sn;                    /* SN of the packet, for media only */

  BOOLEAN mask[24];          /* Mask, FEC only */
  int snbase;                /* SN Base, FEC only */

  struct packet_s *next;

} packet;



BOOLEAN A[65535];
packet *pending_list;

packet *recover_with_fec(packet *fec_pkt) {

  packet *data_pkt;
  int pkts_present,  /* number of packets from the mask that are
                        present */
    pkts_needed,    /* number of packets needed is the number of ones
                        in the mask minus 1 */
    pkt_to_recover, /* sn of the packet we are recovering */
    i;

  pkts_present = 0;

  /* The number of packets needed is the number of ones in the mask
     minus 1.  The code below increments pkts_needed by the number
     of ones in the mask, so we initialize this to -1 so that the
     final count is correct */

  pkts_needed = -1;

  /* Go through all 24 bits in the mask, and check if we have
     all but one of the media packets */

  for(i = 0; i < 24; i++) {

     /* If the packet is here and in the mask, increment counter */

     if(A[i+fec_pkt->snbase] && fec_pkt->mask[i]) pkts_present++;

     /* Count the number of packets needed as well */
     if(fec_pkt->mask[i]) pkts_needed++;
```

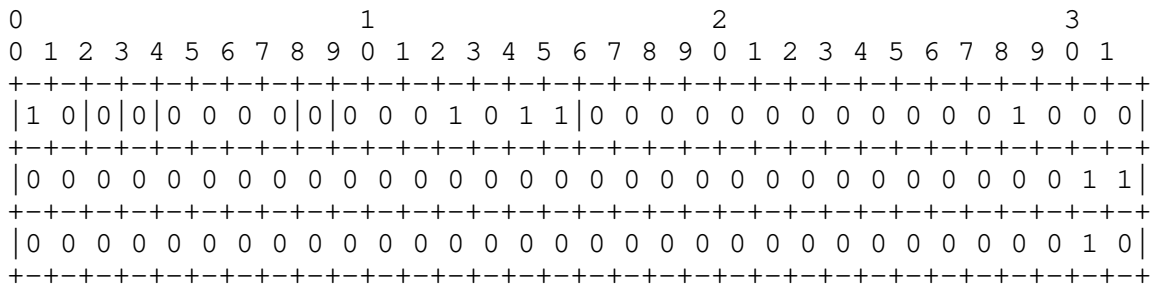
```
    /* The packet to recover is the one with a bit in the
       mask that's not here yet */
    if(!A[i+fec_pkt->snbase] && fec_pkt->mask[i])
      pkt_to_recover = i+fec_pkt->snbase;
  }

  /* If we can recover, do so. Otherwise, return NULL */

  if(pkts_present == pkts_needed) {
    data_pkt = recover_packet(pkt_to_recover, fec_pkt);
  }  else {
    data_pkt = NULL;
  }

  return(data_pkt);
}


void fec_recovery() {

  packet *p,    /* packet received or regenerated */
      *fecp,    /* fec packet from pending list */
      *pnew;    /* new packets recovered */

  while(1) {

    p = wait_for_packet();    /* get packet from network */

    while(p) {

      /* if it's an FEC packet, try to recover with it. If we can't,
         store it for later potential use. If we can recover, act as
         if the recovered packet is received and try to recover some
         more.  Otherwise, if it's a data packet, mark it as received,
         and check if we can now recover a data packet with the list
         of pending FEC packets */

      if(p->fec == TRUE) {
         pnew = recover_with_fec(p);

         if(pnew)

           A[pnew->sn] = TRUE;
         else
           add_fec_to_pending_list(p);

         /* We assign pnew to p since the while loop will continue
            to recover based on p not being NULL */
```

```
          p = pnew;

        } else {

          /* Mark this data packet as here */
          A[p->sn] = TRUE;

          free(p);
          p = NULL;

          /* Go through pending list. Try and recover a packet using
             each FEC. If we are successful, add the data packet to
             the list of received packets, remove the FEC packet from
             the pending list, since we've used it, and then try to
             recover some more */

          for(fecp = pending_list; fecp != NULL; fecp = fecp->next) {
            pnew = recover_with_fec(fecp);
            if(pnew) {

              /* The packet is now here, as we've recovered it */
              A[pnew->sn] = TRUE;

              /* One FEC packet can only be used once to recover,
                 so remove it from the pending list */

              remove_fec_from_pending_list(fecp);

              p = pnew;

              break;
            }

          } /*for*/

        } /*p->fec was false */

      } /* while p*/

    } /* while 1 */

  }
```

9 Example

Consider 2 media packets to be sent, x and y, from SSRC 2. Their sequence numbers are 8 and 9, respectively, with timestamps of 3 and 5, respectively. Packet x uses payload type 11, and packet y uses payload type 18. Packet x is has 10 bytes of payload, and packet y 11. Packet y has its marker bit set. The RTP headers for packets x and y are shown in Figures 3 and 4 respectively.

Media Packet x



Version: 2
Padding: 0
Extension: 0
Marker: 0
PTI: 11
SN: 8
TS: 3
SSRC: 2

Figure 3: RTP Header for Media Packet X

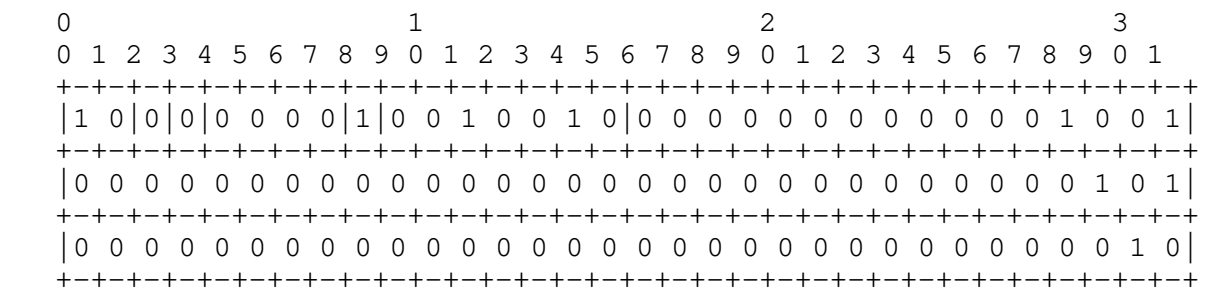
An FEC packet is generated from these two. We assume that payload type 127 is used to indicate an FEC packet. The resulting RTP header is shown in Figure 5.

The FEC header in the FEC packet is shown in Figure 6.

11 Use with Redundant Encodings

One can consider an FEC packet as a "redundant coding" of the media.

Media Packet y



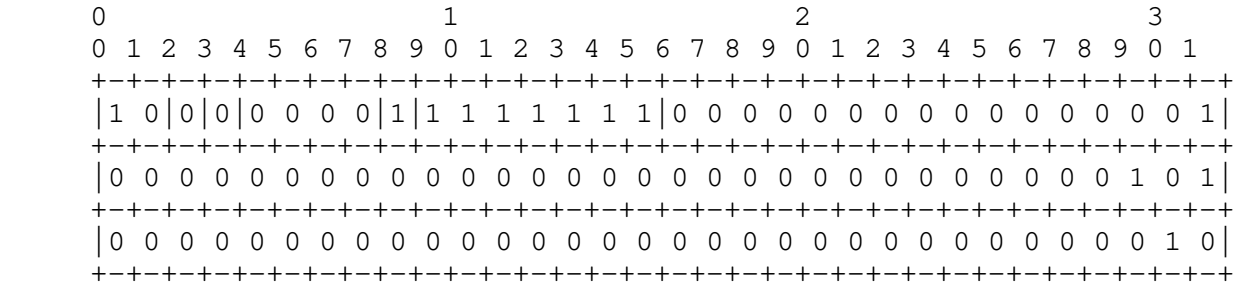

```
Version:   2
Padding:   0
Extension: 0
Marker:    1
PTI:       18
SN:        9
TS:        5
SSRC:      2
```

Figure 4: RTP Header for Media Packet Y

Because of this, the payload format for encoding of redundant audio data [5] can be used to carry the FEC data along with the media. The procedure for this is as follows.

The FEC operation defined above acts on a stream of RTP media packets. The stream which is operated on is the stream before the encapsulation defined in RFC 2198 [5]. In other words, the media stream to be protected is encapsulated in standard RTP media packets. The FEC operation above is performed (with one minor change), generating a stream of FEC packets. The change to the procedure above is that if the RTP packets being protected contain an RTP extension, padding, or a CSRC list, these MUST be removed from the packets, and the CC field, Padding Bit, and Extension but MUST be set to zero, before the FEC operation is applied. These modified packets are used in the procedure above. Note that the sender MUST still send the original packets (with the CSRC list, padding, and extension in tact) as the primary encoding in RFC 2198. The removal of these fields only applies to the protection operation.

Once the FEC packets have been generated, the media payload is extracted from the media packets. This payload is used as the primary encoding as defined in RFC 2198. Then, the FEC header and payload of the FEC packets is extracted, and treated as a redundant encoding. Additional redundant encodings, besides FEC, MAY be added to the packet as well. These encodings will not be protected by FEC, however.



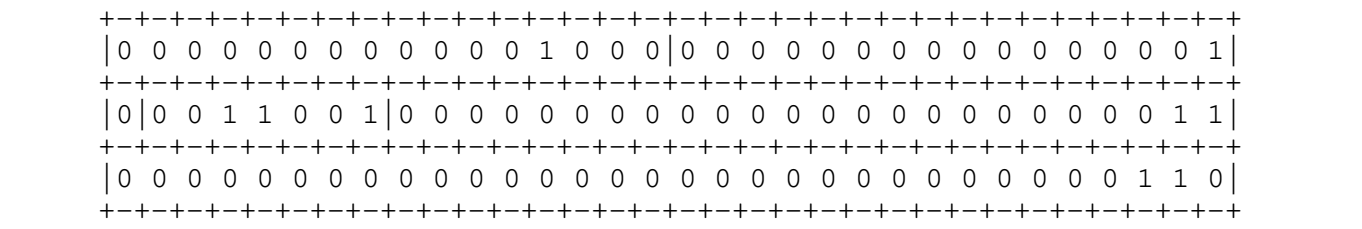
Version: 2
Padding: 0
Extension: 0
Marker: 1
PTI: 127
SN: 1
TS: 5
SSRC: 2

Figure 5: RTP Header of FEC for Packets X and Y

The redundant encodings header for the primary codec is set as defined in RFC 2198. The redundant encodings header for the FEC data is set as follows. The block PT is set to the dynamic PT associated with the FEC format. The block length is set to the sum of the lengths of the FEC header and payload. The timestamp offset SHOULD be set to zero. The secondary coder payload includes the FEC header and FEC payload.

At the receiver, the primary codec and all secondary codecs are extracted as separate RTP packets. This is done by copying the sequence number, SSRC, marker bit, CC field, RTP version, and extension bit from the RTP header of the redundant encodings packet to the RTP header of each extracted packet. If the secondary codec contains FEC, the CC field, Extension Bit, and Padding Bit in the RTP header of the FEC packet MUST be set to zero instead. The payload type identifier in the extracted packet is copied from the block PT of the redundant encodings header. The timestamp of the extracted packet is the difference between the timestamp in the RTP header and

the offset in the block header. The payload of the extracted packet is the data block. This will result in the FEC stream and media stream being extracted.



SN base: 8 [min(8,9)]
len. rec.: 1 [8 xor 9]
E: 0
PTI rec.: 25 [11 xor 18]
mask: 3
TS rec.: 6 [3 xor 5]

The payload length is 11 bytes.

Figure 6: FEC Header of Result

To use the FEC and media packets for recovery, the CSRC list, extension, and padding MUST be removed from the media packets, if present, and the CC field, Extension Bit, and Padding Bit MUST be set to zero. These modified media packets, along with the FEC packets, are then used to recover based on the procedures in section 8. The recovered media packets will always have no extension, padding, or CSRC list. An implementation MAY copy these fields into the recovered packet from another media packet, if available.

Using the redundant encodings payload format also implies that the marker bit may not be recovered correctly. Applications MUST set the marker bit to zero in media packets reconstructed using FEC encapsulated in RFC 2198 redundancy.

An advantage of this approach is a reduction in the overhead for sending FEC packets.

11 Indicating FEC Usage in SDP

FEC packets contain RTP packets with dynamic payload type values. In addition, the FEC packets can be sent on separate multicast groups or separate ports from the media. The FEC can even be carried in packets containing media, using the redundant encodings payload format [5]. These configuration options must be indicated out of band. This section describes how this can be accomplished using the Session Description Protocol (SDP), specified in RFC 2327 [6].

11.1 FEC as a Separate Stream

In the first case, the FEC packets are sent as a separate stream. This can mean they are sent on a different port and/or multicast group from the media. When this is done, several pieces of information must be conveyed:

- The address and port where the FEC is being sent to
- The payload type number for the FEC
- Which media stream the FEC is protecting

The payload type number for the FEC is conveyed in the m line of the media it is protecting, listed as if it were another valid encoding for the stream. There is no static payload type assignment for FEC, so dynamic payload type numbers MUST be used. The binding to the number is indicated by an rtpmap attribute. The name used in this binding is "parityfec".

The presence of the payload type number in the m line of the media it is protecting does not mean the FEC is sent to the same address and port as the media. Instead, this information is conveyed through an fmtp attribute line. The presence of the FEC payload type on the m line of the media serves only to indicate which stream the FEC is protecting.

The format for the fmtp line for FEC is:

a=fmtp:<number> <port> <network type> <addresss type> <connection address>

where 'number' is the payload type number present in the m line. Port is the port number where the FEC is sent to. The remaining three items – network type, address type, and connection address – have the same syntax and semantics as the c line from SDP. This allows the fmtp line to be partially parsed by the same parser used on the c

lines. Note that since FEC cannot be hierarchically encoded, the <number of addresses> parameter MUST NOT appear in the connection address.

The following is an example SDP for FEC:

```
v=0
o=hamming 2890844526 2890842807 IN IP4 126.16.64.4
s=FEC Seminar
c=IN IP4 224.2.17.12/127
t=0 0
m=audio 49170 RTP/AVP 0 78
a=rtpmap:78 parityfec/8000
a=fmtp:78 49172 IN IP4 224.2.17.12/127
m=video 51372 RTP/AVP 31 79
a=rtpmap:79 parityfec/8000
a=fmtp:79 51372 IN IP4 224.2.17.13/127
```

The presence of two m lines in this SDP indicates that there are two media streams - one audio and one video. The media format of 0 indicates that the audio uses PCM, and is protected by FEC with payload type number 78. The FEC is sent to the same multicast group and TTL as the audio, but on a port number two higher (49172). The video is protected by FEC with payload type number 79. The FEC appears on the same port as the video (51372), but on a different multicast address.

11.2 Use with Redundant Encodings

When the FEC stream is being sent as a secondary codec in the redundant encodings format, this must be signaled through SDP. To do this, the procedures defined in RFC 2198 are used to signal the use of redundant encodings. The FEC payload type is indicated in the same fashion as any other secondary codec. An rtpmap attribute MUST be used to indicate a dynamic payload type number for the FEC packets. The FEC MUST protect only the main codec. In this case, the fmtp attribute for the FEC MUST NOT be present.

For example:

```
m=audio 12345 RTP/AVP 121 0 5 100
a=rtpmap:121 red/8000/1
a=rtpmap:100 parityfec/8000
a=fmtp:121 0/5/100
```

This SDP indicates that there is a single audio stream, which can consist of PCM (media format 0) , DVI (media format 5), the redundant encodings (indicated by media format 121, which is bound to red through the rtpmap attribute), or FEC (media format 100, which is bound to parityfec through the rtpmap attribute). Although the FEC format is specified as a possible coding for this stream, the FEC MUST NOT be sent by itself for this stream. Its presence in the m line is required only because non-primary codecs must be listed here according to RFC 2198. The fmtp attribute indicates that the redundant encodings format can be used, with DVI as a secondary coding and FEC as a tertiary encoding.

11.3 Usage with RTSP

RTSP [7] can be used to request FEC packets to be sent as a separate stream. When SDP is used with RTSP, the Session Description does not include a connection address and port number for each stream. Instead, RTSP uses the concept of a "Control URL". Control URLs are used in SDP in two distinct ways.

1. There is a single control URL for all streams. This is referred to as "aggregate control". In this case, the fmtp line for the FEC stream is omitted.

2. There is a Control URL assigned to each stream. This is referred to as "non-aggregate control". In this case, the fmtp line specifies the Control URL for the stream of FEC packets. The URL may be used in a SETUP command by an RTSP client.

The format for the fmtp line for FEC with RTSP and non-aggregate control is:

a=fmtp:<number> <control URL>

where 'number' is the payload type number present in the m line. Control URL is the URL used to control the stream of FEC packets. Note that the Control URL does not need to be an absolute URL. The rules for converting a relative Control URL to an absolute URL are given in RFC 2326, Section C.1.1.

12 Security Considerations

The use of FEC has implications on the usage and changing of keys for encryption. As the FEC packets do consist of a separate stream, there are a number of permutations on the usage of encryption. In particular:

- The FEC stream may be encrypted, while the media stream is not.
- The media stream may be encrypted, while the FEC stream is not.
- The media stream and FEC stream are both encrypted, but using different keys.
- The media stream and FEC stream are both encrypted, but using the same key.

The first three of these would require any application level signaling protocols to be aware of the usage of FEC, and to thus exchange keys for it and negotiate its usage on the media and FEC streams separately. In the final case, no such additional mechanisms are needed. The first two cases present a layering violation, as FEC packets should really be treated no differently than other RTP packets. Encrypting just one may also make certain known-plaintext attacks possible. For these reasons, applications utilizing encryption SHOULD encrypt both streams.

However, the changing of keys becomes problematic. For example, if two packets a and b are sent, and FEC packet f(a,b) is sent, and the keys used for a and b are different, which key should be used to decode f(a,b)? In general, old keys will likely need to be cached, so that when the keys change for the media stream, the old key is kept, and used, until it is determined that the key has changed on the FEC packets as well.

Another issue with the use of FEC is its impact on network congestion. Adding FEC in the face of increasing network losses is a bad idea, as it can lead to increased congestion and eventual congestion collapse if done on a widespread basis. As a result, implementers MUST NOT substantially increase the amount of FEC in use as network losses increase.

13 Acknowledgments

This work is based on an earlier draft on FEC, submitted by Budge and Mackenzie in 1997. We would also like to thank Steve Casner, Mark Handley, Orion Hodson and Colin Perkins for their comments. Thanks to Anders Klemets who wrote the section on usage with RTSP.

14 Authors' Addresses

Jonathan Rosenberg
dynamicsoft
200 Executive Drive
Suite 120
West Orange, NJ 07046

Email: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401, 1214 Amsterdam Ave.
New York, NY 10027-7003

EMail: schulzrinne@cs.columbia.edu

15 Bibliography

[1] J.C. Bolot and A. V. Garcia, "Control mechanisms for packet audio in the internet," in Proceedings of the Conference on Computer Communications (IEEE Infocom) , (San Francisco, California), Mar. 1996.

[2] Perkins, C. and O. Hodson, "Options for Repair of Streaming media", RFC 2354, June 1998.

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

[4] Bradner, S., "Key words for use in RFCs to indicate requirement levels", BCP 14, RFC 2119, March 1997.

[5] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J.C., Vega-Garcia, A. and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[6] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[7] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

16. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.