An RTP Payload Format for Generic Forward Error Correction

1 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.

2 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
different mechanisms for transporting the FEC packets.

3 Terminology

The following terms are used throughout this document:

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

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

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

4. 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 pack-
   ets called FEC packets. The FEC packets are formatted
   according to the rules specified in this document.

5. FEC Header: The FEC header is the header information con-
   tained in an FEC packet.

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

7. 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].

4 Basic Operation

The FEC packets are sent as a separate stream from the media packets. This implies that 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. The media packet stream is essentially unaffected by the use of FEC. This allows the two to be sent on a separate multicast group, so that non-FEC receivers can ignore the FEC and just receive the original media. The separation also allows for coherent values for the sequence numbers and timestamps.

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, arriving FEC and media packets are used to generate a stream of media packets for direct use by the application. This results in a clean separation of error protection from the application.

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

5 Parity Codes

For brevity, we define the function f(x,y,...) to be the XOR (parity) operator applied to the data blocks x,y,... Each data block is simply a set of bits of length L. The function is only well defined when the lengths of the data blocks it operates on are equal. The output of this function is a single data block equal in length to the inputs, called the parity block. The parity block is the bitwise XOR of the input blocks. Note that f(x) = x.

Recovery of data blocks using parity codes is accomplished by generating one or more parity blocks over a group of k packets. To be effective, the parity blocks must be generated by linearly
independent combinations of data blocks. The particular combination is called a parity code. After the parity operation, there will be a total of n data plus parity blocks (i.e., n-k parity blocks). There are a large number of possible parity codes for a given n,k. Reasonable codes exist for large ranges of n and k. The payload format does not mandate a particular code.

For example, consider a parity code which generates a single parity block over two data blocks. The stream of blocks generated by the code is thus:

\[ a, b, f(a,b), c, d, f(c,d) \]

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 letters on the left represent the stream of input data blocks, and the right represents the stream of data and parity blocks.

**Scheme 0**
-------

This scheme is null, and has no error correction. The scheme is formally defined as:

\[ a,b,c,d, ... \rightarrow a, b, c, d, ... \]

**Scheme 1**
-------

This scheme is similar to the one in the example above. The switching of the positions of f(b) and f(a,b) allow some bursts of two consecutive packet losses to be recovered. It is defined as:

\[ a,b,c,d,e,f \rightarrow a, f(a,b), b, f(b,c), c, f(c,d), d, ... \]

**Scheme 2**
-------

This scheme allows for recovery of all single packet losses and some consecutive packet losses, but with less overhead than scheme 1:
a, b, c, d, e, f, g -> f(a, b), f(a, c), f(a, b, c), f(c, d), f(c, e), f(c, d, e)...

Scheme 3
-------

This scheme requires 4 packet delays to recover the original media payloads, but it can recover from 1, 2, or 3 consecutive packet losses:

a, b, c, d, e, f -> f(a), f(b), f(a, b, c), f(c), f(a, c, d), f(a, b, d), f(d), ...

The FEC protocol takes the media payloads as data blocks, and uses the XOR operator on them to generate parity blocks which are the FEC payloads. The xor operator is also applied to portions of the media headers to assist in their recovery.

In order to decode the FEC payloads to media payloads, all that is necessary is for the receiver to know the set of media payloads to which it is applied. This is exactly the information provided by the payload format.

To determine which packets are associated with the FEC packet, a field is present in the FEC header, called the offset mask. Assume this mask is M 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 sequence number base in the FEC packet header. This effectively allows an FEC packet to be associated with any of the M packets before it.

The offset mask and payload type are sufficient to signal arbitrary parity based forward error correction schemes with little overhead.

6 RTP Media Packet Structure

The media packets and FEC packets are sent as separate streams. The media packets are unaffected by FEC, and are sent in the same fashion they would be sent if there were no FEC.

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.

7 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:
7.1 RTP Header

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 is one higher than the sequence number in the previously transmitted FEC packet. The timestamp is set in the following fashion. When the FEC packet is sent, the value of the media RTP timestamp is examined. This value is used as the timestamp of the FEC packet. 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. End systems 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.

7.2 Parity Header

This header is 96 bits. The format of the header is shown in Figure 2.
The length recovery field is used to determine the length of any recovered packets. It is computed via the protection operation applied to the 16 bit natural binary representation 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 for 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 SHOULD 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 CSRC List plus payload plus padding plus extension of the media packets associated with the FEC packet.

8 Protection Operation

The protection operation involves taking a variety of fields from the various headers, appending the payloads, appending the whole thing together, padding zeroes, and then computing the FEC across the resulting binary block. The result is then placed into the FEC packet.

For each media packet to be protected, a binary array is generated by appending the following fields together:

- Padding Bit (1 bit)
- Extension Bit (1 bit)
- CC bits (3 bits)
- Marker bit (1 bit)
- Payload Type (7 bits)
- Timestamp (32 bits)
- Natural binary representation of the length of the CSRC List plus padding plus payload plus extension of the media packet (16 bits)
- CSRC List (if CC is 1), else null (variable)
- Header Extension (if X is 1), else null (variable)
- the payload (variable)
- Padding, if present (variable)

If the lengths of the binary arrays are not equal, they are padded with zeroes to be the length of the longest binary array.

The parity operation is then applied across the binary arrays. The result is the binary array of the FEC packet.

The first bit in the FEC packet binary array is written into the Padding Bit of the FEC packet. The second bit in the FEC packet binary array is written into the Extension bit of the FEC packet. The next
three bits of the FEC packet binary array are written into the CC field of the FEC packet. The next bit of the FEC packet binary array is written into the marker bit of the FEC packet. The next 7 bits of the FEC packet binary array are written into the PT recovery field in the FEC packet header. The next 32 bits of the FEC packet binary array 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.

9 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 can be based on any algorithm chosen by the implementor. Different algorithms result in a tradeoff between complexity and the ability to recover missing packets if at all possible.

9.1 Reconstruction

Let T be the list of packets (FEC and media) which can be combined to recover some media packet xi. For parity, this is an FEC packet plus all but one of the media packets associated with the FEC packet. The procedure is as follows:

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

2. For the FEC packet in T, compute the binary array in the same fashion, except always set the CSRC list, extension, and padding to null.

3. If the resulting binary arrays are not of equal length, pad them with zeroes to be the length of the longest binary array.

4. Perform the exclusive or (parity) operation across the binary arrays, resulting in a recovery array.
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 array.

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

9. Set the CC field to the next three bits in the recovery array.

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

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

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 array.

14. Take the next 16 bits of the recovery array. Whatever the natural binary number this corresponds to, take that many bytes from the recovery array 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 its protecting.

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

9.2 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 implementor. 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 its media. When its an FEC packet, the mask and snbase field contain those values from the FEC packet header. When its 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.

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 thats 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);
    free(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 its 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 its 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 */

                    J.Rosenberg,H.Schulzrinne
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 */

10 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 x 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.

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.
Medi a Packet x

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

Figure 3: Packet X

Medi a Packet y

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

Figure 4: Packet Y
Version: 2
Padding: 0
Extension: 0
Marker: 1
PTI: 127
SN: 1
TS: 5
SSRC: 2

Figure 5: RTP Header of Result

SN base: 8 [min(8,9)]
len. rec.: 1 [8 xor 9]
E: 0
PTI rec.: 24 [11 xor 18]
mask: 3
TS rec.: 6 [3 xor 5]

The payload length is 11 bytes.

Figure 6: FEC Header of Result

11 Use with Redundant Encodings
One can consider an FEC packet as a "redundant coding" of the media. 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 simple. In some media packet, the payload type is set to the value for redundant encodings. The secondary coder is then set to be the FEC data payload type. The block length of the secondary coder is set to the length of the FEC header and payload. The timestamp offset is set to the difference between the media timestamp and the timestamp from the FEC packet. The secondary coder payload includes the FEC header and FEC payload.

This procedure only works if an FEC packet is sent after the last of the media packets it is associated with has been sent. Otherwise, the timestamp offset would be negative, which is not allowed.

Using the redundant encodings payload format also implies that the marker bit cannot be recovered.

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

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. Applications utilizing encryption SHOULD encrypt both streams, however. Encrypting just one may make certain known-plaintext attacks possible.
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, implementors MUST NOT substantially increase the amount of FEC in use as network losses increase.

13 Acknowledgments

The authors would like to thank Budge and Mackenzie, who submitted the initial draft on FEC, and upon which this work is based. We would also like to thank Steve Casner, Orion Hodson and Colin Perkins for their comments.

14 Full Copyright Statement

Copyright (C) The Internet Society (1998). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works.

However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANDABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

15 Author’s Addresses

Jonathan Rosenberg
Lucent Technologies, Bell Laboratories
101 Crawfords Corner Rd.
Holmdel, NJ 07733
Rm. 4C-526
email: jdrosen@bell-labs.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

16 Bibliography


