[Text version]

RMT Working Group
Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-rmt-bb-fec-01.txt 13 July 2000
Expires 13 January 2000

M.Luby/Digital Fountain
L.Vicisano/Cisco
L.Rizzo/U. of Pisa
J.Gemmell/Microsoft
J.Crowcroft/UCL
B. Lueckenhoff/Cadence

Reliable Multicast Transport Building Block:
Forward Error Correction Codes <draft-ietf-rmt-bb-fec-01.txt>

Status of this Memo

Abstract

This memo describes the use of Forward Error Correction (FEC) codes within the context of reliable IP multicast transport and provides an introduction to some commonly-used FEC codes.

1. Rationale and Overview

There are many ways to provide reliability for transmission protocols. A common method is to use ARQ, automatic request for retransmission. With ARQ, receivers use a back channel to the sender to send requests for retransmission of lost packets. ARQ works well for one-to-one reliable protocols, as evidenced by the pervasive success of TCP/IP. ARQ has also been an effective reliability tool for one-to-many reliability protocols, and in particular for some reliable IP multicast protocols. However, for one-to-very many reliability protocols, ARQ has limitations, including the feedback implosion problem because many receivers are transmitting back to the sender, and the need for a back channel to send these requests from the receiver. Another limitation is that receivers may experience different loss patterns of packets, and thus receivers may be delayed by retransmission of packets that other receivers have lost that but they have already received. This may also cause wasteful use of bandwidth used to retransmit packets that have already been received by many of the receivers.

In environments where ARQ is either costly or impossible because there is either a very limited capacity back channel or no back channel at all, such as satellite transmission, a Data Carousel approach to reliability is sometimes used [AFZ95]. With a Data Carousel, the sender partitions the object into equal length pieces of data, which we hereafter call source symbols, places them into packets, and then continually cycles through and sends these packets. Receivers continually receive packets until they have received a copy of each packet. Data Carousel has the advantage that it requires no back channel because there is no data that flows from receivers to the sender. However, Data Carousel also has limitations. For example, if a receiver loses a packet in one round of transmission it must wait an entire round before it has a chance to receive that packet again. This may also cause wasteful use of bandwidth, as the sender continually cycles through and transmits the packets until no receiver is missing a packet.

FEC codes provide a reliability method that can be used to augment or replace other reliability methods, especially for one-to-many reliability protocols such as reliable IP multicast. We first briefly review some of the basic properties and types of FEC codes before reviewing their uses in the context of reliable IP multicast. Later, we provide a more detailed description of some of FEC codes.

In the general literature, FEC refers to the ability to overcome both erasures (losses) and bit-level corruption. However, in the case of IP

multicast, lower network layers will detect corrupted packets and discard them. Therefore, an IP multicast protocol need not be concerned with corruption; the focus is solely on erasure codes. The payloads are generated and processed using an FEC erasure encoder and objects are reassembled from reception of packets using the corresponding FEC erasure decoder.

The input to an FEC encoder is some number k of equal length source symbols. The FEC encoder generates some number of encoding symbols that are of the same length as the source symbols. The chosen length of the symbols can vary upon each application of the FEC encoder, or it can be fixed. These encoding symbols are placed into packets for transmission. The number of encoding symbols placed into each packet can vary on a per packet basis, or a fixed number of symbols (often one) can be placed into each packet. Also, in each packet is placed enough information to identify the particular encoding symbols carried in that packet. Upon receipt of packets containing encoding symbols, the receiver feeds these encoding symbols into the corresponding FEC decoder to recreate an exact copy of the k source symbols. Ideally, the FEC decoder can recreate an exact copy from any k of the encoding symbols.

In a later section, we describe a technique for using FEC codes as described above to handle blocks with variable length source symbols.

Block FEC codes work as follows. The input to a block FEC encoder is k source symbols and a number n. The encoder generates n-k redundant symbols yielding an encoding block of n encoding symbols in total composed of the k source symbols and the n-k redundant symbols. A block FEC decoder has the property that any k of the n encoding symbols in the encoding block is sufficient to reconstruct the original k source symbols.

Expandable FEC codes work as follows. An expandable FEC encoder takes as input k source symbols and generates as many unique encoding symbols as requested on demand, where the amount of time for generating each encoding symbol is the same independent of how many encoding symbols are generated. Unlike block FEC codes, the source symbols are not considered part of the encoding symbols for an expandable FEC code. An expandable FEC decoder has the property that any k of the unique encoding symbols is sufficient to reconstruct the original k source symbols.

Along a different dimension, we classify FEC codes loosely as being either small or large. A small FEC code is efficient in terms of processing time requirements for encoding and decoding for small values

of k, and a large FEC code is efficient for encoding and decoding for much large values of k. There are implementations of standard block FEC codes that have encoding times proportional to n times the length of the k source symbols, and decoding times proportional l times the length of the k source symbols, where l is the number of missing source symbols among the k received encoding symbols. Because of the growth rate of the encoding and decoding times as a function of k and n, these are typically considered to be small block FEC codes. There are close approximations to block FEC codes which for all practical purposes can generate n encoding symbols and can decode the k source symbols in time proportional to the length of the n encoding symbols. These codes are considered to be large block FEC codes. There are close approximations to expandable FEC codes which for all practical purposes can generate each encoding symbol in time proportional to its length, and can decode all k source symbols in time proportional to their length. These are considered to be large expandable FEC codes.

Ideally, FEC codes in the context of IP multicast can be used to generate encoding symbols that are transmitted in packets in such a way that each received packet is fully useful to a receiver to reassemble the object regardless of previous packet reception patterns. Thus, if some packets are lost in transit between the sender and the receiver, instead of asking for specific retransmission of the lost packets or waiting till the packets are resent using Data Carousel, the receiver can use any other subsequent equal amount of data contained in packets that arrive to reassemble the object. These packets can either be proactively transmitted or they can be explicitly requested by receivers. This implies that the data contained in the same packet is fully useful to all receivers to reassemble the object, even though the receivers may have previously experienced different packet loss patterns. This property can reduce or even eliminate the problems mentioned above associated with ARQ and Data Carousel and thereby dramatically increase the scalability of the protocol to orders of magnitude more receivers.

1.1. Application of FEC codecs

For some reliable IP multicast protocols, FEC codes are used in conjunction with ARQ to provide reliability. For example, a large object could be partitioned into a number of source blocks consisting of a small number of source symbols each, and in a first round of transmission all of the source symbols for all the source blocks could be transmitted. Then, receivers could report back to the sender the number of source symbols they are missing from each source block. The

sender could then compute the maximum number of missing source symbols from each source block among all receivers. Based on this, a small block FEC encoder could be used to generate for each source block a number of redundant symbols equal to the computed maximum number of missing source symbols from the block among all receivers. In a second round of transmission, the server would then send all of these redundant symbols for all blocks. In this example, if there are no losses in the second round of transmission then all receivers will be able to recreate an exact copy of each original block. In this case, even if different receivers are missing different symbols in different blocks, transmitted redundant symbols for a given block are useful to all receivers missing symbols from that block in the second round.

For other reliable IP multicast protocols, FEC codes are used in a Data Carousel fashion to provide reliability, which we call an FEC Data Carousel. For example, an FEC Data Carousel using a large block FEC code could work as follows. The large block FEC encoder produces n encoding symbols considering all the k source symbols of an object as one block. The sender cycles through and transmits the n encoding symbols in packets in the same order in each round. An FEC Data Carousel can have much better protection against packet loss than a Data Carousel. For example, a receiver can join the transmission at any point in time, And, as long as the receiver receives at least k encoding symbols during the transmission of the next n encoding symbols, the receiver can completely recover the object. This is true even if the receiver starts receiving packets in the middle of a pass through the encoding symbols. This method can also be used when the object is partitioned into blocks and a short block FEC code is applied to each block separately. In this case, as we explain in more detail below, it is useful to interleave the symbols from the different blocks when they are transmitted.

Since any number of encoding symbols can be generated using an expandable FEC encoder, reliable IP multicast protocols that use expandable FEC codes generally rely solely on these codes for reliability. For example, when an expandable FEC code is used in a FEC Data Carousel application, the encoding packets never repeat, and thus

any k of the encoding symbols in the potentially unbounded number of encoding symbols are sufficient to recover the original k source symbols.
For yet other reliable IP multicast protocols the method to obtain reliability is to generate enough encoding symbols so that each encoding symbol is transmitted at most once. For example, the sender can decide a priori how many encoding symbols it will transmit, use an FEC code to

generate that number of encoding symbols from the object, and then transmit the encoding symbols to all receivers. This method is for example applicable to streaming protocols, where the stream is partitioned into objects, the source symbols for each object are encoded into encoding symbols using an FEC code, and then the sets of encoding symbols for each object are transmitted one after the other using IP multicast.

2. FEC Codes

2.1. Simple codes

There are some very simple codes that are effective for repairing packet loss under very low loss conditions. For example, one simple way to provide protection from a single loss is to partition the object into fixed size source symbols and then add a redundant symbol that is the parity (XOR) of all the source symbols. The size of a source symbol is chosen so that it fits perfectly into the payload of a packet, i.e. if the packet payload is 512 bytes then each source symbol is 512 bytes. The header of each packet contains enough information to identify the payload. In this case, this includes a symbol ID. The symbol IDs are numbered consecutively starting from zero independently for the source symbols and for the redundant symbol. Thus, the packet header also contains an encoding flag that indicates whether the symbol in the payload is a source symbol or a redundant symbol, where 1 indicates source symbol and 0 indicates redundant symbol. For example, if the object consists of four source symbols that have values a, b, c and d, then the value of the redundant symbol is e = a XOR b XOR c XOR d. Then, the packets carrying these symbols look like

(0, 1: a), (1, 1: b), (2, 1: c), (3, 1: d), (0, 0: e).

In this example, the first two fields are in the header of the packet, where the first field is the symbol ID and the second field is the encoding flag. The portion of the packet after the colon is the payload. Any single symbol of the object can be recovered as the parity of all the other symbols. For example, if packets

(0, 1: a), (1, 1: b), (3, 1: d), (0, 0: e)

are received then the symbol value for the missing source symbol with ID 2 can be recovered by computing a XOR b XOR d XOR e = c.

When the number of source symbols in the object is large, a simple block code variant of the above can be used. In this case, the source symbols are grouped together into source blocks of some number k of consecutive symbols each, where k may be different for different blocks. If a block consists of k source symbols then a redundant symbol is added to form an encoding block consisting of k+1 encoding symbols. Then, a source block consisting of k source symbols can be recovered from any k of the k+1 encoding symbols from the associated encoding block.
Slightly more sophisticated ways of adding redundant symbols using parity can also be used. For example, one can group a block consisting of k source symbols in an object into a p x p square matrix, where p = sqrt(k). Then, for each row a redundant symbol is added that is the parity of all the source symbols in the row. Similarly, for each column a redundant symbol is added that is the parity of all the source symbols in the column. Then, any row of the matrix can be recovered from any p of the p+1 symbols in the row, and similarly for any column. Higher dimensional product codes using this technique can also be used. However, one must be wary of using these constructions without some thought towards the possible loss patterns of symbols. Ideally, the property that one would like to obtain is that if k source symbols are encoded into n encoding symbols (the encoding symbols consist of the source symbols and the redundant symbols) then the k source symbols can be recovered from any k of the n encoding symbols. Using the simple constructions described above does not yield codes that come close to obtaining this ideal behavior.

2.2. Small block FEC codes

Reliable IP multicast protocols may use a block (n, k) FEC code [BLA84]. A popular example of these types of codes is a class of Reed-Solomon codes. For such codes, k source symbols are encoded into n > k encoding symbols, such that any k of the encoding symbols can be used to reassemble the original k source symbols. Thus, these codes have 0% reception overhead when used to encode the entire object directly. Block codes are usually systematic, which means that the n encoding symbols consist of the k source symbols and n-k redundant symbols generated from these k source symbols, where the size of a redundant symbol is the same as that for a source symbol. For example, the first simple code (XOR) described in the previous subsection is a (k+1, k) code. In general, the freedom to choose n larger than k+1 is desirable, as this can provide much better protection against losses. Codes of this sort are often based on algebraic methods using finite fields. Some of the most popular such codes are based on linear block codes.

Implementations of (n, k) FEC erasure codes are efficient enough to be used by personal computers [RIZ97c, NON97]. For example, [RIZ97b] describes an implementation where the encoding and decoding speeds decay as C/j, where the constant C is on the order of 10 to 80 Mbytes/second for Pentium class machines of various vintages and j is upper bounded by min(k, n-k).
In practice, the values of k and n must be small (below 256) for such FEC codes as large values make encoding and decoding prohibitively expensive. As many objects are longer than k symbols for reasonable values of k and the symbol length (e.g. larger

than 16 kilobyte for k = 16 using 1 kilobyte symbols), they can be divided into a number of source blocks. Each source block consists of some number k of source symbols, where k may vary between different source blocks. The FEC encoder is used to encode a k source symbol source block into a n encoding symbol encoding block, where the length n of the encoding block may vary for each source block. For a receiver to completely recover the object, for each source block consisting of k source symbols, k distinct encoding symbols (i.e., with different symbol IDs) must be received from the corresponding encoding block. For some encoding blocks, more encoding symbols may be received than there are source symbols in the corresponding source block, in which case any additional encoding symbols are discarded. An example encoding structure is shown in Figure 1.


```
        |   source symbols        |   source symbols        |
        |   of source block 0     |   of source block 1     |
                     |                           |
                     v                           v
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
        |0 |1 |2 |3 |4 |5 |6 |7 |0 |1 |2 |3 | 4|5 |6 |7 |
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
                                 |
                            FEC encoder
                                 |
                                 v
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |0 |1 |2 |3 | 4| 5| 6| 7| 8| 9| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
                   ^                              ^
                   |                              |
    |  encoding symbols             | encoding symbols              |
    |  of encoding block 0          | of encoding block 1           |
```

Figure 1. Encoding structure for object divided into two source blocks consisting of 8 source symbols each, and the FEC encoder is used to generate 2 additional redundant symbols (10 encoding symbols in total) for each of the two source blocks.

In many cases, an object is partitioned into equal length source blocks each consisting of k contiguous source symbols of the object, i.e., block c consists of the range of source symbols [ck, (c+1)k-1]. This ensure that the FEC encoder can be optimized to handle a particular number k of source symbols. This also ensures that memory references are local when the sender reads source symbols to encode, and when the receiver reads encoding symbols to decode. Locality of reference is particularly important when the object is stored on disk, as it reduces the disk seeks required. The block number and the source symbol ID within that block can be used to uniquely specify a source symbol within the object. If the size of the object is not a multiple of k source symbols, then the last source block will contain less than k symbols.
Encoding symbols can be uniquely identified by block number and encoding symbol ID. The block numbers can be numbered consecutively starting from zero. One way of identifying encoding symbols within a block are to use symbol IDs and an encoding flag that is used to specify whether an encoding symbol is a source symbol or a redundant symbol, where for example 1 indicates source symbol and 0 indicate redundant symbol. The

symbol IDs can be numbered consecutively starting from zero for each block independently for the source symbols and for the redundant symbols. Thus, an encoding symbol can be identified by its block number, the encoding flag, and the symbol ID. For example, if the object consists 10 source symbols with values a, b, c, d, e, f, g, h, i, and j, and k = 5 and n = 8, then there are two source blocks consisting of 5 symbols each, and there are two encoding blocks consisting of 8 symbols each. Let p, q and r be the values of the redundant symbols for the first encoding block, and let x, y and z be the values of the redundant symbols for the second encoding block. Then the encoding symbols together with their identifiers are

```
(0, 0, 1: a), (0, 1, 1: b), (0, 2, 1: c), (0, 3, 1: d), (0, 4, 1: e),
(0, 0, 0: p), (0, 1, 0: q), (0, 2, 0: r),
(1, 0, 1: f), (1, 1, 1: g), (1, 2, 1: h), (1, 3, 1: i), (1, 4, 1: j),
(1, 0, 0: x), (1, 1, 0: y), (1, 2, 0: z).
```

In this example, the first three fields identify the encoding symbol, where the first field is the block number, the second field is the symbol ID and the third field is the encoding flag. The value of the encoding symbol is written after the colon. Each block can be recovered from any 5 of the 8 encoding symbols associated with that block. For example, reception of (0, 1, 1: b), (0, 2, 1: c), (0, 3, 1: d), (0, 0, 0: p) and (0, 1, 0: q) are sufficient to recover the first source block and reception of (1, 0, 1: f), (1, 1, 1: g), (1, 0, 0: x), (1, 1, 0: y) and (1, 2, 0: z) are sufficient to recover the second source block.

2.3. Large block FEC codes

Tornado codes [LUB97] are block FEC codes that provide an alternative to small block FEC codes. A (n, k) Tornado code requires slightly more than k out of n encoding symbols to reassemble k source symbols. However, the advantage is that the value of k may be on the order of tens of thousands and still run efficiently. Because of memory considerations, in practice the value of n is restricted to be a small multiple of k, e.g., n = 2k. For example, [BYE98] describes an implementation of Tornado codes where the encoding and decoding speeds are tens of megabytes per second range for Pentium class machines of various vintages when k is in the tens of thousands and n = 2k. The reception overhead for such values of k and n is in the 5-10% range. Tornado codes require a large amount of out of band information to be communicated to all senders and receivers for each different object

Draft RMT BB, Forward Error Correction Codes 13 July 2000

length, and require an amount of memory on the encoder and decoder which is proportional to the object length times 2n/k.

Tornado codes are designed to have low reception overhead on average with respect to reception of a random portion of the encoding packets. Thus, to ensure that a receiver can reassemble the object with low reception overhead, the packets are permuted into a random order before transmission.

2.4. Expandable FEC codes

All of the FEC codes described up to this point are block codes. There is a different type of FEC codes that we call expandable FEC codes. Like block codes, an expandable FEC encoder operates on an object of known size that is partitioned into equal length source symbols. Unlike block codes, ideally there is no predetermined number of encoding symbols that can be generated for expandable FEC codes. Instead, an expandable FEC encoder can generate as few or as many unique encoding symbols as required on demand. Also unlike block codes, optimal expandable FEC codes have the additional attractive property that encoding symbols for the same object can be generated and transmitted from multiple servers and concurrently received by a receiver and yet the receiver incurs a 0% reception overhead.

LT codes [LUB00] are an example of large expandable FEC codes. An LT encoder uses randomization to generate each encoding symbol randomly and independently of all other encoding symbols. Like Tornado codes, the number of source symbols k may be very large for LT codes, i.e., on the order of tens to hundreds of thousands, and the encoder and decoder run efficiently in software. For example the encoding and decoding speeds for LT codes are in the range 3-20 megabytes per second for Pentium class machines of various vintages when k is in the high tens of thousands. An LT encoder closely approximates the properties of an ideal expandable FEC encoder, as it can generate as few or as many encoding symbols as required on demand. When a new encoding symbol is to be generated by an LT encoder, it is based on a randomly chosen 32-bit encoding symbol ID that uniquely describes how the encoding symbol is to be generated from the input symbols. In general, each encoding symbol ID value corresponds to a unique encoding symbol, and thus the space of possible encoding symbols is approximately four billion. Thus, the chance that a particular encoding symbol is the same as any other particular encoding symbol is tiny. An LT decoder has the property that with very high probability the receipt of any set of slightly more than k randomly and independently generated encoding

symbols is sufficient to reassemble the k source symbols. For example, when k is on the order of tens to hundreds of thousands the reception overhead is less than 5% with no failures in tens of millions of trials under a variety of loss conditions.

Because encoding symbols are randomly and independently generated by choosing random encoding symbol IDs, LT codes have the property that encoding symbols for the same k source symbols can be generated and transmitted from multiple senders ad than if all the encoding symbols were generated by a single sender. The only requirement is that the senders choose their encoding symbol IDs randomly and independently of one another.

There is a weak tradeoff between the number of source symbols and the reception overhead for LT codes, and the larger the number of source symbols the smaller the reception overhead. Thus, for shorter objects, it is sometimes advantageous to include multiple symbols in each packet. Normally, and in the discussion below, there is only one symbol per packet.

There are a couple of factors for choosing the appropriate symbol length/ number of input symbols tradeoff. The primary consideration is that there is a fixed overhead per symbol component in the overall processing requirements of the encoding and decoding, independent of the number of input symbols. Thus, using shorter symbols means that this fixed overhead processing per symbol will be a larger component of the overall processing requirements, leading to larger overall processing requirements. Because of this, it is advisable to use a reasonably sized fixed symbol length independent of the length of the object, and thus shorter objects will be partitioned into fewer symbols than larger objects. A second much less important consideration is that there is a component of the processing per symbol that depends logarithmically on the number of input symbols, and thus for this reason there is a slight preference towards less input symbols.

Like small block codes, there is a point when the object is large enough that it makes sense to partition it into blocks when using LT codes. Generally the object is partitioned into blocks whenever the number of source symbols times the packet payload length is less than the size of the object. Thus, if the packet payload is 1024 bytes and the number of source symbols is 64,000 then any object over 64 megabytes will be partitioned into more than one block. One can choose the number of source symbols to partition the object into, depending on the desired encoding and decoding speed versus reception overhead tradeoff desired. Encoding symbols can be uniquely identified by a block number (when the

object is large enough to be partitioned into more than one block) and an encoding symbol ID. The block numbers, if they are used, are generally numbered consecutively starting from zero within the object. The block number and the encoding symbol ID are both chosen uniformly and randomly from their range when an encoding symbol is to be generated and transmitted. For example, suppose the number of source symbols is 64,000 and the number of blocks is 2. Then, each packet generated by the LT encoder could be of the form (b, x: y). In this example, the first two fields identify the encoding symbol, where the first field is the block number b = 0 or 1 and the second field is the randomly chosen encoding symbol ID x. The value y after the colon is the value of the encoding symbol.

2.5. Source blocks with variable length source symbols

For all the FEC codes described above, all the source symbols in the same source block are all of the same length. In this section, we describe a general technique to handle the case when it is desirable to use source symbols of varying lengths in a single source block. This technique is applicable to block FEC codes.

Let l_1, l_2, ... , l_k be the lengths of k varying length source symbols to be considered part of the same source block. Let lmax be the maximum over i = 1, ... , k of l_i. To prepare the source block for the FEC encoder, pad each source symbol i out to length lmax with a suffix of lmax-i zeroes, and then prepend to the beginning of this the value l_i. Thus, each padded source symbol is of length x+lmax, assuming that the length of an original symbol takes x bytes to store. These padded source symbols, each of length x+lmax, are the input to the encoder, together with the value n. The encoder then generates n-k redundant symbols, each of length x+lmax.

The encoding symbols that are placed into packets consist of the original k varying length source symbols and n-k redundant symbols, each of length x+lmax. >From any k of the received encoding symbols, the FEC decoder recreates the k original source symbols as follows. If all k original source symbols are received, then no decoding is necessary. Otherwise, at least one redundant symbol is received, from which the receiver can easily whether the block was composed of variable-length source symbols: if the redundant symbol(s) has a size different (larger) from the source symbols then the source symbols are variable-length. Note that in a variable-length block the redundant symbols are always larger than the largest source symbol, due to the presence of the encoded symbol-length. The receiver can determine the value of lmax by

subtracting x from the length of a received redundant symbol. Note that x MUST be a protocol constant. For each of the received original source symbols, the receiver can generate the corresponding padded source symbol as described above. Then, the input to the FEC decoder is the set of received redundant symbols, together with the set of padded source symbols constructed from the received original symbols. The FEC decoder then produces the set of k padded source symbols. Once the k padded source symbols have been recovered, the length l_i of original source symbol i can be recovered from the first x bytes of the ith padded source symbol, and then original source symbol i is obtained by taking the next l_i bytes following the x bytes of the length field.

3. FEC Abstract Packet Fields and Out-of-Band Information

This section specify the information that protocol packets must carry to implement the various forms of FEC-based reliability. A session is defined to be all the information associated with a transmission of data about a particular object by a single sender. There are three classes of packets that may contain FEC information within a session: data packets, session-control packets and feedback packets. They generally contain different kinds of FEC information. Note that some protocols do not use feedback packets.
Data packets MAY sometime serve as session-control packets as well; both data and session-control packets generally travel downstream (from the sender towards receivers) and are addressed to a multicast IP address (sometime the might be addressed to the unicast address of a specific receiver). In the following, for simplicity we will refer to both data and session control packets as downstream-traveling packets, or simply downstream packets.
As a general rule, feedback packets travel upstream (from receivers to the sender) and are addressed to the unicast address of the sender. Sometimes, however, they might be addressed to a multicast IP address or to the unicast address of a receiver or also the the unicast address of some different node (intermediate node that provides recovery services or neighboring router).

The FEC-related information that can be present in downstream packets can be classified as follows:

```
 1) FEC Encoding Identifier
```

```
       Identifies the FEC encoding being used and has the purpose of
       allowing receivers to select the appropriate FEC decoder. As a
       general rule, the "FEC Encoding Identifier" MUST be the same for
       a given session, i.e., for all transmission of data related to a
       particular object, but MAY vary across different transmissions of
       data about different objects in different sessions, even if
       transmitted using the same set of multicast groups.
 2) FEC payload ID
       Identifies the symbol(s) in the payload of the packet. The
       content of this piece of information depends on the encoder being
       used (e.g. in Block FEC codes this may be the combination of
       block index and symbol index; in expandable FEC codes this may be
       just a flat symbol identifier).
 3) FEC Object Transmission Information
       This is information regarding the encoding of a specific object
       needed by the FEC decoder (e.g. for Block FEC codes this may be
       the combination of block length and object length). This might
       also include general parameters of the FEC encoder.
```

All the classes of information above, except 2), can either be transmitted within the transport session (using protocol packet-header fields) or out of band. The information described in 2) MUST be transmitted in data-packet header fields, as it provides a description of the data contained in the packet. In the following we specify the content of 1), 2) and 3) independent of whether these pieces of information are transmitted in protocol packets or out of band. This document does not specify out of band methods to transport the information.
Within the context of FEC repair schemes, feedback packets are (optionally) used to request FEC retransmission. The FEC-related information present in feedback packets can be classified as follow:

```
 1) FEC Block Identifier
       This is the identifier of the FEC block for which retransmission
       is requested. This information does not apply to some type of
       decoders.
```

```
 2) Number of Repair Symbols
       This is the number of repair symbols requested, needed to recover
       the object.
```

3.1. FEC Encoding Identifier

This is a numeric index that identifies a specific FEC encoding scheme OR a class of encoding schemes that share the same format of "FEC Payload ID" and "FEC Object Transmission Information".
The FEC Encoding Identifier identifies a specific FEC encoding scheme when the encoding scheme is formally and fully specified, in a way that independent implementors can implement both encoder and decoder from the specification. Companion documents of this specification may specify such FEC encoding schemes and associate them with "FEC Encoding Identifier" values. These documents MUST also specify a correspondent format for the "FEC Payload ID" and "FEC Object Transmission Information". Currently FEC Encoding Identifiers in the range 0-127 are reserved for this class of encoding schemes.

It is possible that a FEC encoding scheme cannot be completely specified or that such a specification is simply not available or also that a party exists that owns the encoding scheme and it is not willing to disclose its algorithm. We refer to these encoding schemes as "Under- Specified" schemes. Under-specified schemes can still be identified as follows:

```
 o   A format of the fields "FEC Payload ID" and "FEC Object
     Transmission Information" MUST be defined for the encoding scheme.
 o   A value of "FEC Encoding Identifier" MUST be reserved and
     associated to the format definitions above. An already reserved
     "FEC Encoding Identifier"  MUST be reused if it is associated to
     the same format of "FEC Payload ID" and "FEC Object Transmission
     Information" as the ones needed for the new under-specified FEC
     encoding scheme.
```

o A value of "FEC Encoding Name" must be reserved (see below).
An Under-specified FEC scheme is completely identified by the tuple (FEC Encoding Identifier, FEC Encoding Name). The party that owns this tuple MUST be able to provide an FEC encoder and decoder that implement the

under-specified FEC encoding scheme identified by the tuple.

"FEC Encoding Names" are numeric identifiers scoped by a FEC Encoding Identifier.

The FEC Encoding Name MUST be part of the "FEC Object Transmission Information" and must be communicated to receivers together with the FEC Encoding Identifier.
An FEC Encoding Identifier MAY also define a format for the (abstract) feedback packet fields "FEC Block Identifier" and "Number of Repair Symbols".

3.2. FEC Payload ID and FEC Object Transmission Information

A document that specifies an encoding scheme and reserves a value of FEC Encoding Identifier MUST define a packet-field format for FEC Payload ID and FEC Object Transmission Information according to the need of the encoding scheme. This also applies to documents that reserves values of FEC Encoding Identifiers for under-specified encoding schemes. In this case the FEC Object Transmission Information must also include a field to contain the "FEC Encoding Name".
A packet field definition of FEC Object Transmission Information MUST be provided despite the fact that protocol instantiation may decide to communicate this information out of band.
The packet field format of "FEC Block Identifier" and "Number of Repair Symbols" SHOULD be specified for each FEC encoding scheme, even the scheme is mainly intended for feedback-less protocols. FEC Block Identifier may not apply to some encoding schemes.
All packet field definition (FEC Payload ID, FEC Object Transmission Information, FEC Block Identifier and Number of Repair Symbols) MUST be fully specified at

level of bit-fields and they must have a length that is a multiple of a 4-byte word (this is to facilitate the alignment of packet fields in protocol instantiations).

4. IANA Considerations

Values of FEC Encoding Identifiers and FEC Encoding Names are subject to IANA registration. FEC Encoding Identifiers and FEC Encoding Names are hierarchical: FEC Encoding Identifiers (at the top level) scope ranges

of FEC Encoding Names. Not all FEC Encoding Identifiers have a corresponding FEC Encoding Name scope (see below).

A FEC Encoding Identifier is a numeric non-negative index. Value from 0 to 127 are reserved for FEC encoders that are fully specified, as described in section 3.1. The assignment of a FEC Encoding Identifier in this range can only be granted if the requestor can provide such a specification published as an IETF RFC. Value grater than 127 can be assigned to under-specified encoding schemes.
In any case values of FEC Encoding Identifiers can only be assigned if the required FEC packet fields corresponding to it (see section 3.1) are specified in a RFC.
Each FEC Encoding Identifier assigned to an under-specified encoding scheme scopes a range of FEC Encoding Names. An FEC Encoding Name is a numeric non-negative index. The document that reserves a FEC Encoding Identifier MAY also specify a range for the subordinate FEC Encoding Names.
Under the scope of a FEC Encoding Identifier, FEC Encoding Names are assigned on a First Come First Served base to requestors that are able to provide point of contact information and a pointer to publicly accessible documentation describing the FEC encoder and a ways to obtain it. The requestor is responsible for keeping this information up to date.

5. Security Considerations

The use of FEC, in and of itself, imposes no additional security considerations versus sending the same information without FEC. However, just like for any transmission system, a malicious sender may intentionally transmit bad symbols. If a receiver accepts one or more bad symbols in place of authentic ones then such a receiver will have its entire object down-load corrupted by the bad symbol. Application- level transmission object authentication can detect the corrupted transfer, but the receiver must then discard the transferred object. Thus, transmitting false symbols is at least an effective denial of service attack. At worst, a malicious sender could add, delete, or replace arbitrary data within the transmitted object.
In light of this possibility, FEC receivers may screen the source address of a received symbol against a list of authentic transmitter addresses. Since source addresses may be spoofed, FEC transport

protocols may provide mechanisms for robust source authentication of each encoded symbol. Multicast routers along the path of a FEC transfer may provide the capability of discarding multicast packets that originated on that subnet, and whose source IP address does not correspond with that subnet.

6. Intellectual Property Disclosure

Tornado codes [LUB97] have both patents issued and patents pending. LT codes [LUB00] have patents pending.

7. Acknowledgments

Thanks to Vincent Roca and Hayder Radha for their detailed comments on this draft.

8. References

[AFZ95] Acharya, S., Franklin, M., and Zdonik, S., ‘‘Dissemination- Based Data Delivery Using Broadcast Disks’’, IEEE Personal Communications, pp.50-60, Dec 1995.

[BLA94] Blahut, R.E., ‘‘Theory and Practice of Error Control Codes’’, Addison Wesley, MA 1984.

[BYE98] Byers, J.W., Luby, M., Mitzenmacher, M., and Rege, A., ‘‘A Digital Fountain Approach to Reliable Distribution of Bulk Data’’, Proceedings ACM SIGCOMM ’98, Vancouver, Canada, Sept 1998.

[DEE88] Deering, S., ‘‘Host Extensions for IP Multicasting’’, RFC 1058, Stanford University, Stanford, CA, 1988.

[GEM99] Gemmell, J., Schooler, E., and Gray, J., ‘‘ALC Scalable Multicast File Distribution: Caching and Parameters Optimizations’’ Technical Report MSR-TR-99-14, Microsoft Research, Redmond, WA, April, 1999.

[HAN98] Handley, M., and Jacobson, V., ‘‘SDP: Session Description Protocol’’, RFC 2327, April 1998.

[HAN96] Handley, M., ‘‘SAP: Session Announcement Protocol’’, Internet Draft, IETF MMUSIC Working Group, Nov 1996.

[LUB97] Luby, M., Mitzenmacher, M., Shokrollahi, A., Spielman, D., Stemann, V., ‘‘Practical Loss-Resilient Codes’’ 29th STOC’97.

[LUB99] Luby, M., Vicisano, L., Speakman, T. ‘‘Heterogeneous multicast congestion control based on router packet filtering'', presented at RMT meeting in Pisa, March 1999.

[LUB00] Luby, M., "An overview of LT codes", Digital Fountain white paper, July 2000.

[R2068] Fielding, R., Gettys, J., Mogul, J. Frystyk, H., Berners-Lee, T., Hypertext Transfer Protocol HTTP/1.1 (IETF RFC2068) http://www.rfc-editor.org/rfc/rfc2068.txt

[R2119] Bradner, S., Key words for use in RFCs to Indicate Requirement Levels (IETF RFC 2119) http://www.rfc-editor.org/rfc/rfc2119.txt

[RIZ97a] Rizzo, L, and Vicisano, L., ''Reliable Multicast Data Distribution protocol based on software FEC techniques'', Proceedings of the Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems, HPCS-97, Chalkidiki, Greece, June 1997.

[RIZ97b] Rizzo, L., and Vicisano, L., ''Effective Erasure Codes for Reliable Computer Communication Protocols'', ACM SIGCOMM Computer Communication Review, Vol.27, No.2, pp.24-36, Apr 1997.

[RIZ97c] Rizzo, L., ''On the Feasibility of Software FEC'', DEIT Tech Report, http://www.iet.unipi.it/ luigi/softfec.ps, Jan 1997.

[RUB99] Rubenstein, Dan, Kurose, Jim and Towsley, Don, ''The Impact of Multicast Layering on Network Fairness'', Proceedings of ACM SIGCOMM'99.

[VIC98A] L.Vicisano, L.Rizzo, J.Crowcroft, ''TCP-like Congestion Control for Layered Multicast Data Transfer'', IEEE Infocom '98, San Francisco, CA, Mar.28-Apr.1 1998.

[VIC98B] Vicisano, L., ''Notes On a Cumulative Layered Organization of Data Packets Across Multiple Streams with Different Rates'', University College London Computer Science Research Note RN/98/25, Work in Progress (May 1998).

A. Predefined FEC encoders

This appendix specifies the FEC Encoding Identifier and the relative packets field for a number of known schemes that follow under the class of under-specified FEC encoding schemes. Others may be specified in companion documents.

A.1. Small Block, Large Block and Expandable FEC Codes

This section reserves a FEC Encoding Identifier for the family of codes described in Section 2.2, 2.3 and 2.4 and specifies the relative packet fields. Under-specified FEC encoding schemes that belong to this class MUST use this identifier and packet field definitions.

The FEC Encoding Identifier assigned to Small Block, Large Block, and Expandable FEC Codes is 128.

The FEC Payload ID is composed of an encoding symbol index and an encoding block number structured 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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     encoding block number                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                      encoding symbol ID                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In addition, a one bit FEC Encoding Flag SHOULD be included, and this flag indicates whether the encoding symbol(s) in the payload of the packet are source symbol(s) or redundant symbol(s). The FEC Object Transmission Information has the following structure:


```
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      FEC Encoding Name        |      Object Length (MSB)      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     Object Length (LSB)                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     Source Block Length                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note that this structure limits the range of possible FEC Encoding Names to 0-:-65536, despite the FEC Object Transmission Information can also be transmitted out of band.

The Object Length, composed of a Most Significant Bytes portion (MSB) and a Least Significant Bytes portion (LSB), is expressed in bytes.

Feedback packet MUST contain both FEC Block Identifier and Number of Repair Symbols. Their structure is the following:

```
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                      FEC Block Identifier                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                   Number of Repair Symbols                    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

9. Authors' Addresses

Michael Luby
luby@digitalfountain.com
Digital Fountain
600 Alabama Street
San Francisco, CA, USA, 94110

Lorenzo Vicisano
lorenzo@cisco.com
cisco Systems, Inc.
170 West Tasman Dr.,
San Jose, CA, USA, 95134
Luigi Rizzo
luigi@iet.unipi.it
Dip. di Ing. dell'Informazione
Universita` di Pisa
via Diotisalvi 2, 56126 Pisa, Italy
Jim Gemmell
jgemmell@microsoft.com
Microsoft Research
301 Howard St., #830
San Francisco, CA, USA, 94105
Jon Crowcroft
J.Crowcroft@cs.ucl.ac.uk
Department of Computer Science
University College London
Gower Street,
London WC1E 6BT, UK
Bruce Lueckenhoff
brucelu@cadence.com
Cadence Design Systems, Inc.
120 Cremona Drive, Suite C
Santa Barbara, CA 93117
Draft RMT BB, Forward Error Correction Codes 13 July 2000

10. Full Copyright Statement

Table of Contents