Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-rmt-info-fec-03.ps

RMT WG
M. Luby/Digital Fountain
L. Vicisano/Cisco
J. Gemmell/Microsoft
L. Rizzo/ACIRI and Univ. Pisa
M. Handley/ACIRI
J. Crowcroft/UCL

3 September 2002
Expires: March 2003

The Use of Forward Error Correction in Reliable Multicast

Status of this Memo

This document is a product of the IETF RMT WG. Comments should be addressed to the authors, or the WG's mailing list at rmt@lbl.gov.

Abstract

This memo describes the use of Forward Error Correction codes to efficiently provide and/or augment reliability for one-to-many reliable data transport using IP multicast. One of the key properties of Forward Error Correction codes in this context is the ability to use the same packets containing Forward Error Correction data to simultaneously repair different packet loss patterns at multiple receivers. Different classes of Forward Error Correction codes and some of their basic properties are described and terminology relevant to implementing Forward Error Correction Codes in a reliable multicast protocol is introduced. Examples are provided of possible abstract formats for packets carrying Forward Error Correction data.

Table of Contents

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 [1]. 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.

Forward Error Correction (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 an IP multicast protocol, the network layers will detect corrupted packets and discard them or the transport layers can use packet authentication to discard corrupted packets. Therefore the primary application of FEC codes to IP multicast protocols is as an erasure code. The payloads are generated and processed using an FEC erasure encoder and objects are reassembled from reception of packets containing the generated encoding 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 a total of n encoding symbols. The encoder is systematic if it 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. An expandable FEC decoder has the property that any k of the unique encoding symbols is sufficient to reconstruct the original k source symbols.

The above definitions explain the ideal situation when the reception of any k encoding symbols is sufficient to recover the k source symbols, in which case the reception overhead is 0%. For some practical FEC codes, slightly more than k encoding symbols are needed to recover the k source symbols. If k*(1+ep) encoding symbols are needed, we say the reception overhead is ep*100%, e.g., if k*1.05 encoding symbols are needed then the reception overhead is 5%.

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 block FEC codes that have encoding times proportional to n-k times the length of the k source symbols, and decoding times proportional to l times the length of the k source symbols, where l is the number of missing source symbols among the k received encoding symbols and l can be as large as k. Because of the growth rate of the encoding and decoding times as a product of k and n-k, these are typically considered to be small block FEC codes. There are block FEC codes with a small reception overhead that 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 expandable FEC codes with a small reception overhead that can generate each encoding symbol in time roughly proportional to its length, and can decode all k source symbols in time roughly proportional to their length. These are considered to be large expandable FEC codes. We describe examples of all of these types of codes later.

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 number of 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 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, as long as this maximum maximum for each block does not exceed the number of redundant symbols that can be generated efficiently. 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 is an encoding symbol ID. The encoding symbol IDs can consist of two parts in this example. The first part is an encoding flag that is equal to 1 if the encoding symbol is a source symbol and is equal to 0 if the encoding symbol is a redundant symbol. The second part of the encoding symbol ID is a source symbol ID if the encoding flag is 1 and a redundant symbol ID if the encoding flag is 0. The source symbol IDs can be numbered from 0 to k-1 and the redundant symbol ID can be 0. 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

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

In this example, the encoding symbol ID consists of the first two values, where the first value is the encoding flag and the second value is either a source symbol ID or the redundant symbol ID. The portion of the packet after the colon is the value of the encoding symbol. Any single source symbol of the object can be recovered as the parity of all the other symbols. For example, if packets

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

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

Another way of forming the encoding symbol ID is to let values 0,...,k-1 correspond to the k source symbols and value k correspond to the redundant symbol that is the XOR of the k source symbols.

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 [2]. 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 no 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. A popular example of these types of codes is a class of Reed-Solomon codes, which are based on algebraic methods using finite fields. Implementations of (n, k) FEC erasure codes are efficient enough to be used by personal computers [16]. For example, [15] 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 (for example 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 number n of encoding symbols in 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 encoding 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 the excess encoding symbols are discarded. An example encoding structure is shown in Figure 1.

```
    |   source symbol IDs   |   source symbols IDs  |
    |   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 symbol IDs         | encoding symbol IDs         |
|  of encoding block 0         | of encoding block 1         |
```

Figure 1. The encoding structure for an 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 ensures 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.

The block numbers can be numbered consecutively starting from zero. Encoding symbols within a block can be uniquely identified by an encoding symbol ID. One way of identifying encoding symbols within a block is to use the combination of an encoding flag that identifies the symbol as either a source symbol or a redundant symbol together with either a source symbol ID or a redundant symbol ID. For example, an encoding flag value of 1 can indicate that the encoding symbol is a source symbol and 0 can indicate that it is a redundant symbol. The source symbol IDs can be numbered from 0 to k-1 and the redundant symbol IDs can be numbered from 0 to n-k-1.

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

In this example, the first value identifies the block number and the second two values together identify the encoding symbol within the block, i.e, the encoding symbol ID consists of the encoding flag together with either the source symbol ID or the redundant symbol ID depending on the value of 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, 1, 2: c), (0, 1, 3: d), (0, 0, 0: p), (0, 0, 1: q)
```

is sufficient to recover the first source block, and reception of

```
(1, 1, 0: f), (1, 1, 1: g), (1, 0, 0: x), (1, 0, 1: y), (1, 0, 2: z)
```

is sufficient to recover the second source block.

Another way of uniquely identifying encoding symbols within a block is to let the encoding symbol IDs for source symbols be 0,...,k-1 and to let the encoding symbol IDs for redundant symbols be k,...,n-1.

2.3. Large block FEC codes

Tornado codes [12], [13], [10], [11], [9] are large block FEC codes that provide an alternative to small block FEC codes. An (n, k) Tornado code requires slightly more than k out of n encoding symbols to recover k source symbols, i.e., there is a small reception overhead. The benefit of the small cost of non-zero reception overhead is that the value of k may be on the order of tens of thousands and still the encoding and decoding are efficient. 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, [4] describes an implementation of Tornado codes where the encoding and decoding speeds are tens of megabytes per second 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 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, 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.

LT codes [6], [7], [8], [5] are an example of large expandable FEC codes with a small reception overhead. 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 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 encoding symbol ID that uniquely describes how the encoding symbol is to be generated from the source 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 if for example the encoding symbol ID is 4 bytes. Thus, the chance that a particular encoding symbol is the same as any other particular encoding symbol is inversely proportional to the number of possible encoding symbol IDs. 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 hundreds of millions of trials under any 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 and received by a receiver and the reception overhead and the decoding time is the same as if though 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 partition the object into many short source symbols and include multiple encoding symbols in each packet. In this case, a single encoding symbol ID is used to identify the multiple encoding symbols contained in a single packet.

There are a couple of factors for choosing the appropriate symbol length/ number of source symbols tradeoff. The primary consideration is that there is a fixed overhead per symbol in the overall processing requirements of the encoding and decoding, independent of the number of source 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. A second much less important consideration is that there is a component of the processing per symbol that depends logarithmically on the number of source symbols, and thus for this reason there is a slight preference towards fewer source 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 maximum number of source symbols is 128,000 then any object over 128 megabytes will be partitioned into more than one block. One can choose the maximum number of source symbols to use, 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 transmitted. For example, suppose the number of source symbols is 128,000 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 value identifies the block number and the second value identifies the encoding symbol within the block. In this example, the block number b is either 0 or 1, and the encoding symbol ID x might be a 32-bit value. 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 in bytes 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-l_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 it takes x bytes to store an integer with possible values 0,...,lmax, where x is a protocol constant known to both the encoder and the decoder. 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 determine if the block is composed of variable-length source symbols: if the redundant symbol(s) is longer than the source symbols then the source symbols are variable-length. Note that in a variable-length block the redundant symbols are always longer than the longest source symbol, due to the presence of the prepended symbol-length. The receiver can determine the value of lmax by subtracting x from the length of a received redundant symbol. 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. 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 try to inject packets carrying corrupted encoding symbols. If a receiver accepts one or more corrupted encoding symbol in place of authentic ones then such a receiver may reconstruct a corrupted object.

Application-level transmission object authentication can detect the corrupted transfer, but the receiver must then discard the transferred object. Thus, injecting corrupted encoding symbols they are accepted as valid encoding symbols by a receiver is at the very least an effective denial of service attack.

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, transport protocols using FEC may provide mechanisms for robust source authentication of each encoding 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.

It is recommended that a packet authentication scheme such as TESLA [14] be used in conjunction with FEC codes. Then, packets that cannot be authenticated can be discarded and the object can be reliably recovered from the received authenticated packets.

4. Intellectual Property Disclosure

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

5. Acknowledgments

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

6. References

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

[2] Blahut, R.E., “Theory and Practice of Error Control Codes”, Addison Wesley, MA 1984.

[3] Bradner, S., "The Internet Standards Process -- Revision 3", RFC2026, October 1996.

[4] 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.

[5] Haken, A., Luby, M., Horn, G., Hernek, D., Byers, J., Mitzenmacher, M., "Generating high weight encoding symbols using a basis", U.S. Patent No. 6,411,223, June 25, 2002.

[6] Luby, M., "Information Additive Code Generator and Decoder for Communication Systems", U.S. Patent No. 6,307,487, October 23, 2001.

[7] Luby, M., "Information Additive Group Code Generator and Decoder for Communication Systems", U.S. Patent No. 6,320,520, November 20, 2001.

[8] Luby, M., "Information Additive Code Generator and Decoder for Communication Systems", U.S. Patent No. 6,373,406, April 16, 2002.

[9] Luby, M., Mitzenmacher, ‘‘Loss resilient code with double heavy tailed series of redundant layers’’, U.S. Patent No. 6,195,777, February 27, 2001.

[10] Luby, M., Mitzenmacher, M., Shokrollahi, A., Spielman, D., Stemann, V., ‘‘Message encoding with irregular graphing’’, U.S. Patent No. 6,163,870, December 19, 2000.

[11] Luby, M., Mitzenmacher, M., Shokrollahi, A., Spielman, D., ‘‘Efficient Erasure Correcting Codes’’, IEEE Transactions on Information Theory, Special Issue: Codes on Graphs and Iterative Algorithms, pp. 569-584, Vol. 47, No. 2, February 2001.

[12] Luby, M., Shokrollahi, A., Stemann, V., Mitzenmacher, M., Spielman, D., ‘‘Loss resilient decoding technique’’, U.S. Patent No. 6,073,250, June 6, 2000.

[13] Luby, M., Shokrollahi, A., Stemann, V., Mitzenmacher, M., Spielman, D., ‘‘Irregularly graphed encoding technique’’, U.S. Patent No. 6,081,909, June 27, 2000.

[14] Perrig, A., Canetti, R., Song, D., Tygar, J.D., "Efficient and Secure Source Authentication for Multicast", Network and Distributed System Security Symposium, NDSS 2001, pp. 35-46, February 2001.

[15] Rizzo, L., ‘‘Effective Erasure Codes for Reliable Computer Communication Protocols’’, ACM SIGCOMM Computer Communication Review, Vol.27, No.2, pp.24-36, Apr 1997.

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

7. Authors’ Addresses

Michael Luby
luby@digitalfountain.com
Digital Fountain
39141 Civic Center Drive
Suite 300
Fremont, CA 94538

Lorenzo Vicisano
lorenzo@cisco.com
cisco Systems, Inc.
170 West Tasman Dr.,
San Jose, CA, USA, 95134

Jim Gemmell
jgemmell@microsoft.com
Microsoft Research
301 Howard St., #830
San Francisco, CA, USA, 94105

Luigi Rizzo
luigi@iet.unipi.it
Dip. di Ing. dell'Informazione
Universita' di Pisa
via Diotisalvi 2, 56126 Pisa, Italy

Mark Handley
mjh@icir.org
ICSI Center for Internet Research
1947 Center St.
Berkeley CA, USA, 94704

Jon Crowcroft
J.Crowcroft@cl.cam.ac.uk
Marconi Professor of Communications Systems
University of Cambridge
Computer Laboratory
William Gates Building
J J Thomson Avenue
Cambridge
CB3 0FD

8. Full Copyright Statement