RMT Working Group   M. Luby
INTERNET DRAFT   L. Vicisano
Expires 8 September 2000   L. Rizzo
                                                            J. Gemmell
                                                          J. Crowcroft
                                                        B. Lueckenhoff
                                                          8 March 2000

Reliable Multicast Transport Building Block: Forward Error Correction Codes

<draft-ietf-rmt-bb-fec-00.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

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 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. Ideally, FEC codes in the context of IP multicast can be used to encode an object into 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 retransmission using ARQ 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. 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.

For some reliable IP multicast protocols, FEC codes are used in conjunction with ARQ to provide reliability. For example, in a first round all of the source symbols could be transmitted, and then receivers could report back to the sender the number of symbols they are missing from each block. Then, the sender could compute the maximum number of missing symbols from each block among all receivers, and then transmit that number of redundant symbols for each 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.

For others, FEC codes are used in a Data Carousel fashion to provide reliability, by cycling through and transmitting the encoding symbols instead of the source symbols. For example, suppose an FEC code is applied to the entire object consisting of k source symbols to generate n encoding symbols with the property that the entire object can be reassembled from any k encoding symbols, and the sender cycles through and transmits the n encoding symbols in the same order in each round. Then, 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 n encoding symbols then the receiver can completely recover the object. This is true even if the receiver joins the data carousel in the middle of a round.

For yet other reliable IP multicast protocols the sole method to obtain reliability is to use FEC codes. 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, each object is 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. The large on demand codes described below have the property that the FEC encoder can generate sequentially as many encoding symbols as are desired on demand. Thus, reliable IP multicast protocols that use large on demand codes generally rely solely on these codes for reliability.

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 primary purpose of using FEC codes is to ensure that minimal number of packets need be received in order for a receiver to reassemble an object. Reception overhead is used to measure how close a protocol comes to achieving this minimum. Reception overhead is the aggregate length of packets needed to recover the object beyond the object length, measured as a percentage of the object length. For example, if it takes 15 MB of packets in order to recover a 10 MB object, then the reception overhead is (15 10)/10 times 100, or 50%. The minimal reception overhead possible is 0%.

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 sym-

bol. Thus, the packet header also contains an encoding flag that indicates whether they 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 k

consecutive symbols each, and then for each block of source symbols a redundant symbol is added to form encoding blocks of k+1 symbols each. Then, a source block can be recovered from any k of the k+1 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 the k source symbols in the 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 codes

Reliable IP multicast protocols may use a block (n, k) FEC erasure code [BLA84]. A popular example of this 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 for these 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 KB for k = 16 using 1 KB symbols), they are divided into m source blocks consisting of k source symbols each. An erasure code is used to encode each source block into an encoding block consisting of n encoding symbols. For a receiver to completely recover the object, k distinct encoding symbols (i.e., with different symbol IDs) must be received for each of the encoding blocks. For some encoding blocks, more than k encoding symbols may be received, in which case any additional encoding symbols are discarded. An example encoding structure is shown in Figure 1.

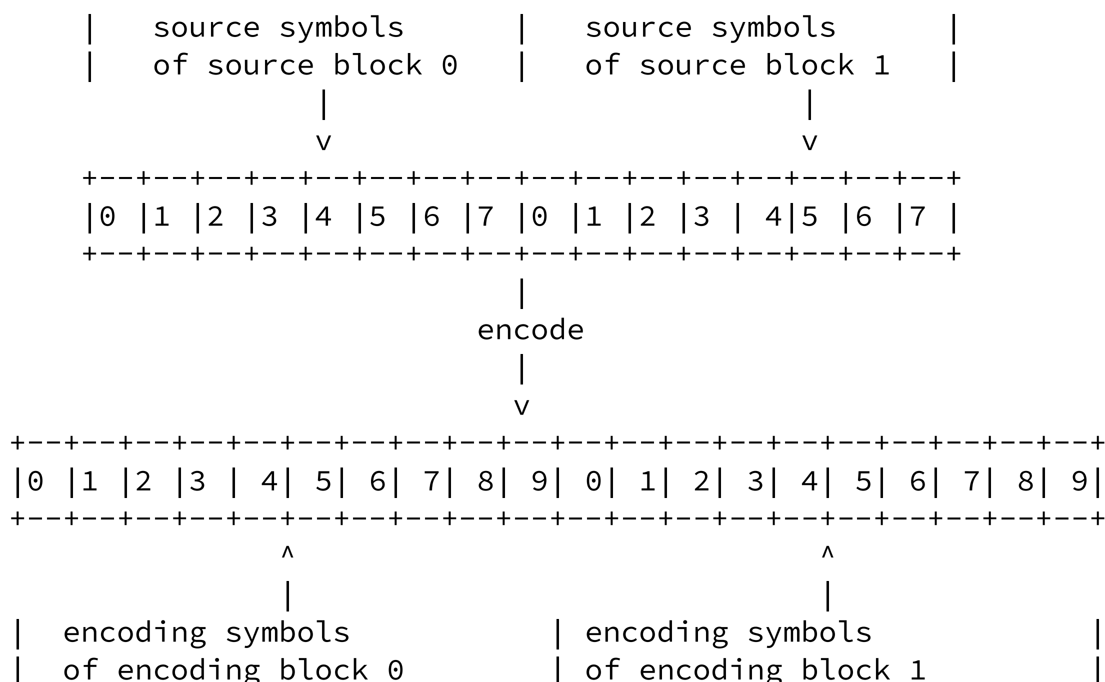

Figure 1. Encoding structure for object divided into m = 2 source blocks, k = 8 and n = 10

When using small block codes for objects that are larger than k source symbols in length, the source symbols in the object are assigned to blocks. Typically, each k contiguous source symbols of the object is assigned to a block, i.e., block c consists of the range of source symbols [ck, (c+1)k-1]. This ensures that memory reference 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) and (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 codes

Tornado codes [LUB97] are block FEC erasure codes that provide an alternative to small block 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 proportional to 10 Mbytes/second to 80 Mbytes/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. On demand codes

All of the FEC erasure codes described up to this point are block codes. There is a different type of FEC erasure code that we call on demand codes. Like block codes, an on demand 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 on demand codes. Instead, an on demand encoder can generate as few or as many encoding symbols as required on demand. Also unlike block codes, optimal on demand 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 are an example of a large on demand FEC code. An LT encoder uses randomization to generate each encoding symbol randomly and independently of all other encoding symbols. An LT encoder satisfies the on demand property, as it can generate as few or as many encoding symbols as required on demand. Let k be the number of source symbols that the object is partitioned into. An LT decoder has the property that with very high probability the receipt of any set of slightly more than k encoding symbols is sufficient to reassemble the object. Like Tornado codes, the value of k may be very large, i.e., on the order of tens or hundreds of thousands, and the encoder and decoder run efficiently in software. For example the encoding and decoding speeds for LT codes are proportional to 3 Mbytes/second to 20 Mbytes/second for Pentium class machines of various vintages when k is in the high tens of thousands. The reception overhead for such values of k is in the 2-4% range.

When a new encoding symbol is to be generated, it is based on a key that uniquely describes how the encoding symbol is to be generated. Because encoding symbols are randomly and independently generated, LT codes have the property that encoding symbols for the same object can

be generated and transmitted from multiple servers and concurrently received by a receiver with no more reception overhead than if all the encoding symbols were generated by a single sender.

There is a tradeoff between the number of source symbols and the reception overhead, 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.

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 MB 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 range of possible values for an encoding symbol ID is orders of magnitude larger than the number of source symbols in a block, i.e., on the range of possible values is generally in the billions. 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.

3. Passing FEC coding information to receivers

There are two basic methods for passing FEC coding information to receivers in order to decode an object: within the IP multicast packet headers or through out of band methods. A description of the variety of out of band methods is outside the scope of this document. The FEC coding information can be classified as three types. FEC session information is information needed by the FEC decoder that may

remain fixed for the transmission of many objects. FEC object transmission information is information particular to the object

transmission session needed by the FEC decoder. The FEC payload ID identifies the symbols in the payload of the ALC packet.

FEC coding information include the FEC codec type, the source block length, the symbol length, the object length, the encoding block number, the encoding symbol ID, and an encoding flag indicating whether the encoding symbol is a source symbol or a redundant symbol. The FEC codec type, the source block length and the symbol length are often FEC session information, although they may classified as FEC object transmission information for some protocols. Thus, sometimes this information is passed to the receiver out of band, although they can equally well be included in each IP multicast packet header as long as the amount of space they take within each packet is minimal. The object length is part of FEC object transmission information. Depending on the protocol, the object length is passed to the receiver out of band or included within each IP multicast packet header. The FEC payload ID consists of the encoding block number (if used), the encoding symbol ID and the encoding flag. The FEC payload ID must be contained within each IP multicast packet header.

4. 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.

5. Intellectual Property Disclosure

Both Tornado codes and LT codes have patents pending.

6. Acknowledgments

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

7. 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.

[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 IEEES 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).

8. Authors' Addresses

Michael Luby
luby@dfountain.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