Effective Erasure Codes for Reliable Computer Communication Protocols

Luigi Rizzo
Dip. di Ingegneria dell'Informazione, Università di Pisa
via Diotisalvi 2 – 56126 Pisa (Italy) – email: `l.rizzo@iet.unipi.it`

Abstract

Reliable communication protocols require that all the intended recipients of a message receive the message intact. Automatic Repeat reQuest (ARQ) techniques are used in unicast protocols, but they do not scale well to multicast protocols with large groups of receivers, since segment losses tend to become uncorrelated thus greatly reducing the effectiveness of retransmissions. In such cases, Forward Error Correction (FEC) techniques can be used, consisting in the transmission of redundant packets (based on error correcting codes) to allow the receivers to recover from independent packet losses.

Despite the widespread use of error correcting codes in many fields of information processing, and a general consensus on the usefulness of FEC techniques within some of the Internet protocols, very few actual implementations exist of the latter. This probably derives from the different types of applications, and from concerns related to the complexity of implementing such codes in software. To fill this gap, in this paper we provide a very basic description of erasure codes, describe an implementation of a simple but very flexible erasure code to be used in network protocols, and discuss its performance and possible applications. Our code is based on Vandermonde matrices computed over $GF(p^r)$, can be implemented very efficiently on common microprocessors, and is suited to a number of different applications, which are briefly discussed in the paper. An implementation of the erasure code shown in this paper is available from the author, and is able to encode/decode data at speeds up to several MB/s running on a Pentium 133.

Keywords: Reliable multicast, FEC, erasure codes.

1 Introduction

Computer communications generally require reliable[1] data transfers among the communicating parties. This is usually achieved by implementing reliability at different levels in the protocol stack, either on a link-by-link basis (e.g. at the link layer), or using end-to-end protocols at the transport layer (such as TCP), or directly in the application.

[0] The work described in this paper has been supported in part by the Commission of European Communities, Esprit Project LTR 20422 – "Moby Dick, The Mobile Digital Companion (MOBYDICK)", and in part by the Ministero dell'Università e della Ricerca Scientifica e Tecnologica of Italy.

[1] Throughout this paper, with reliable we mean that data must be transferred with no errors and no losses.

ARQ (Automatic Repeat reQuest) techniques are generally used in unicast protocols: missing packets are retransmitted upon timeouts or explicit requests from the receiver. When the bandwidth-delay product approaches the sender's window, ARQ might result in reduced throughput. Also, in multicast communication protocols ARQ might be highly inefficient because of uncorrelated losses at different (groups of) receivers.

In these cases, Forward Error Correction (FEC) techniques, possibly combined with ARQ, become useful: the sender prevents losses by transmitting some amount of redundant information, which allow the reconstruction of missing data at the receiver without further interactions. Besides reducing the time needed to recover the missing packets, such an approach generally simplifies both the sender and the receiver since it might render a feedback channel unnecessary; also, the technique is attractive for multicast applications since different loss patterns can be recovered from using the same set of transmitted data.

FEC techniques are generally based on the use of error detection and correction codes. These codes have been studied for a long time and are widely used in many fields of information processing, particularly in telecommunications systems. In the context of computer communications, error detection is generally provided by the lower protocol layers which use checksums (e.g. Cyclic Redundancy Checksums (CRCs)) to discard corrupted packets. Error correcting codes are also used in special cases, e.g. in modems, wireless or otherwise noisy links, in order to make the residual error rate comparable to that of dedicated, wired connections. After such link layer processing, the upper protocol layers have mainly to deal with *erasures*, i.e. missing packets in a stream. Erasures originate from uncorrectable errors at the link layer (but those are not frequent with properly designed and working hardware), or, more frequently, from congestion in the network which causes otherwise valid packets to be dropped due to lack of buffers. Erasures are easier to deal with than errors since the exact position of missing data is known.

Recently, many applications have been developed which use multicast communication. Some of these applications, e.g. audio or videoconferencing tools, tolerate segment losses with a relatively graceful degradation of performance, since data blocks are often independent of each other and have a limited lifetime. Others, such as electronic whiteboards or diffusion of circular information over the network ("electronic newspapers", distribution of software, etc), have instead more strict requirements and require reliable delivery of all data. Thus, they would greatly benefit from an increased reliability in the communication.

Despite an increased need, and a general consensus on their usefulness [4, 10, 14, 19] there are very few Internet protocols which use FEC techniques. This is possibly due to the existence of a gap between the telecommunications world, where FEC techniques have been first studied and developed, and the computer communications world. In the former, the interest is focused on error correcting codes, operating on relatively short strings of bits and implemented on dedicated hardware; in the latter, erasure codes are needed, which must be able to operate on packet-sized data objects, and need to be implemented efficiently in software using general-purpose processors.

In this paper we try to fill this gap by providing a basic description of the principles of operation of erasure codes, presenting an erasure code which is easy to understand, flexible and efficient to implement even on inexpensive architectures, and discussing various issues related to its performance and possible applications. The paper is structured as follows: Section 2 gives a brief introduction to the principles of operation of erasure codes. Section 3 describes our code and discusses some issues related to its implementation on general purpose processors. Finally, Section 4 briefly shows a number of possible applications in computer communication protocols, both in unicast and multicast protocols. A portable C implementation of the erasure code described in this paper is available from the author [16].

2 An introduction to erasure codes

In this section we give a brief introduction to the principle of operation of erasure codes. For a more in-depth discussion of the problem the interested reader is referred to the copious literature on the subject [1, 11, 15, 20]. In this paper we only deal with the so-called *linear block codes* as they are simple and appropriate for the applications of our interest.

The key idea behind erasure codes is that k blocks of *source* data are encoded at the sender to produce n blocks of *encoded* data, in such a way that any subset of k encoded blocks suffices to reconstruct the source data. Such a code is called an (n, k) code and allows the receiver to recover from up to $n - k$ losses in a group of n encoded blocks. Figure 1 gives a graphical representation of the encoding and decoding process.

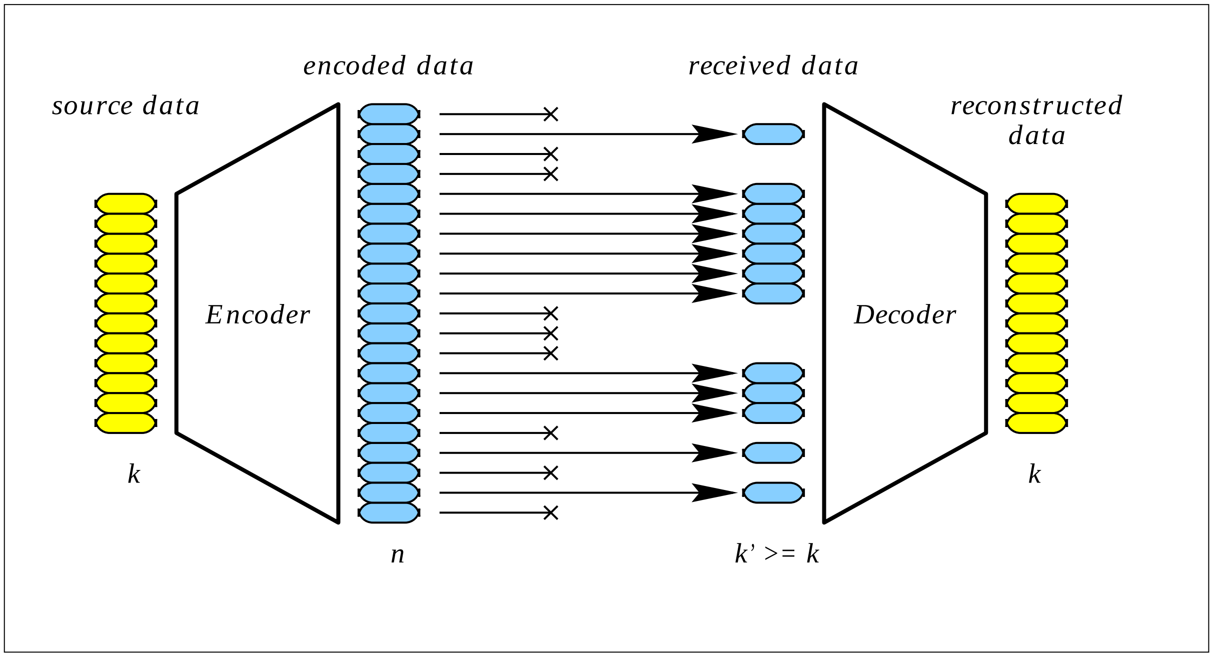

Figure 1: A graphical representation of the encoding/decoding process.

Within the telecommunications world, a block is usually made of a small number of bits. In computer communications, the "quantum" of information is generally much larger – one packet of data, often amounting to hundreds or thousands of bits. This changes somewhat the way an erasure code can be implemented. However, in the following discussion we will assume that a block is a single data item which can be operated on with simple arithmetic operations. Large packets can be split into multiple data items, and the encoding/decoding process is applied by taking one data item per packet.

An interesting class of erasure codes is that of *linear codes*, so called because they can be analyzed using the properties of linear algebra. Let $\underline{x} = x_0 \dots x_{k-1}$ be the source data, G an $n \times k$ matrix, then an (n, k) linear code can be represented by

$$\underline{y} = G\underline{x}$$

for a proper definition of the matrix G. Assuming that k components of $\underline{y}$ are available at the receiver, source data can be reconstructed by using the k equations corresponding to the known components of $\underline{y}$. We call G' the $k \times k$ matrix representing these equations (Figure 2). This of course is only possible if these equations are linearly independent, and, in the general case, this holds if any $k \times k$ matrix extracted from G is invertible.

If the encoded blocks include a verbatim copy of the source blocks, the code is called a *systematic code*. This corresponds to including the identity matrix I_k in G. The advantage of a systematic code is that it simplifies the reconstruction of the source data in case one expects very few losses.

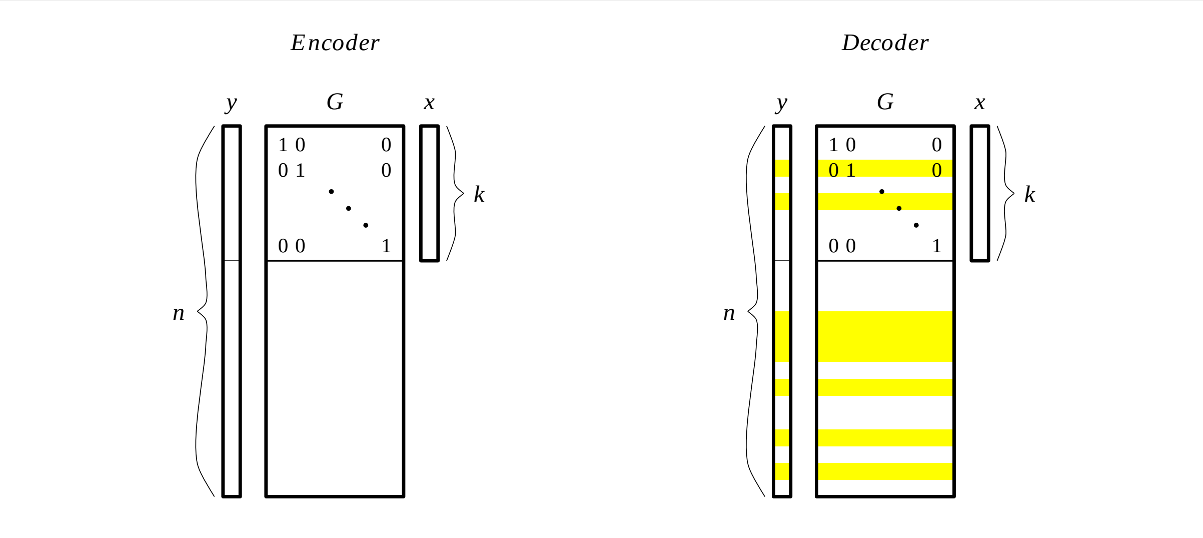

Figure 2: The encoding/decoding process in matrix form, for a systematic code (the top k rows of G constitute the identity matrix I_k). $\underline{y}'$ and G' correspond to the grey areas of the vector and matrix on the right.

2.1 The generator matrix

G is called the *generator* matrix of the code, because any valid $\underline{y}$ is a linear combination of columns of G. Since G is an $n \times k$ matrix with rank k, any subset of k encoded blocks should convey information on all the k source blocks. As a consequence, each column of G can have at most $k-1$ zero elements. In the case of a systematic code G contains the identity matrix I_k, which consumes all zero elements. Thus the remaining rows of the matrix must all contain non-zero elements.

Strictly speaking, the reconstruction process needs some additional information – namely, the identity of the various blocks – to reconstruct the source data. However, this information is generally derived by other means and thus might not need to be transmitted explicitly. Also, in the case of computer communications, this additional information has a negligible size when compared to the size of a packet.

There is however another source of overhead which cannot be neglected, and this is the precision used for computations. If each x_i is represented using b bits, representing the y_i's requires more bits if ordinary arithmetic is used. In fact, if each coefficient g_{ij} of G is represented on b' bits, the y_i's need $b+b'+\lceil \log_2 k \rceil$ bits to be represented without loss of precision. That is a significant overhead, since those excess bits must be transmitted to reconstruct the source data. Rounding or truncating the representation of the y_i's would prevent a correct reconstruction of the source data.

2.2 Avoiding roundings: computations in finite fields

Luckily the expansion of data can be overcome by working in a finite field. Roughly speaking, a *field* is a set in which we can add, subtract, multiply and divide, in much the same way we are used to work on integers (the interested reader is referred to some textbook on algebra [6] or coding theory (e.g. [1, Ch.2 and Ch.4]), where a more formal presentation of finite fields is provided; a relatively simple-to-follow presentation is also given in [2, Chap.2]). A field is *closed* under addition and multiplication, which means that the result of sums and products of field elements are still field elements. A *finite* field is characterized by having a finite number of elements. Most of the properties of linear algebra apply to finite fields as well.

The main advantage of using a finite field, for our purposes, lies in the closure property which allows us to make exact computations on field elements without requiring more bits to represent the results. In order to work on a finite field, we need to map our data elements into field elements, operate upon them according to the rules of the field, and then apply the inverse mapping to reconstruct the desired results.

2.2.1 Prime fields

Finite fields have been shown to exist with $q = p^r$ elements, where p is a prime number. Fields with p elements, with p prime, are called *prime fields* or $GF(p)$, where GF stands for Galois Field. Operating in a prime field is relatively simple, since $GF(p)$ is the set of integers from 0 to $p-1$ under the operations of addition and multiplication modulo p. From the point of view of a software implementation, there are two minor difficulties in using a prime field: first, with the exception of $p=2$, field elements require $\lceil \log_2 p \rceil > \log_2 p$ bits to be represented. This causes a slight inefficiency in the encoding of data, and possibly an even larger inefficiency in operating on these numbers since the operand sizes might not match the word size of the processor. The second problem lies in the need of a modulo operation on sums and, especially, multiplications.

The modulo is an expensive operation since it requires a division. Both problems, though, can be minimized if $p = 2^m + 1$.

2.2.2 Extension fields

Fields with $q = p^r$ elements, with p prime and $r > 1$, are called *extension fields* or $GF(p^r)$. The sum and product in extension fields are *not* done by taking results modulo q. Rather, field elements can be considered as polynomials of degree $r - 1$ with coefficients in $GF(p)$. The sum operation is just the sum between coefficients, modulo p; the product is the product between polynomials, computed modulo an irreducible polynomial (i.e. one without divisors in $GF(p^r)$) of degree r, and with coefficients reduced modulo p.

Despite the apparent complexity, operations on extension fields can become extremely simple in the case of $p = 2$. In this case, elements of $GF(2^r)$ require exactly r bits to be represented, a property which simplifies the handling of data. Sum and subtraction become the same operation (a bit-by-bit sum modulo 2), which is simply implemented with an exclusive OR.

2.2.3 Multiplications and divisions

An interesting property of prime or extension fields is that there exist at least one special element, usually denoted by α, whose powers generate all non-zero elements of the field. As an example, a generator for $GF(5)$ is 2, whose powers (starting from 2^0) are $1, 2, 4, 3, 1, \ldots$. Powers of α repeat with a period of length $q - 1$, hence $\alpha^{q-1} = \alpha^0 = 1$.

This property has a direct consequence on the implementation of multiplication and division. In fact, we can express any non-zero field element x as $x = \alpha^{k_x}$. k_x can be considered as "logarithm" of x, and multiplication and division can be computed using logarithms, as follows:

$$xy = \alpha^{|k_x + k_y|_{q-1}}, \quad \frac{1}{x} = \alpha^{q-1-k_x}$$

where $|a|_b$ stands for "a modulo b". If the number of field elements not too large, tables can be built off line to provide the "logarithm", the "exponential" and the multiplicative inverse of each non-zero field element. In some cases, it can be convenient to provide a table for multiplications as well. Using the above techniques, operations in extension fields with $p = 2$ can be extremely fast and simple to implement.

2.3 Data recovery

Recovery of original data is possible by solving the linear system

$$\underline{y}' = G'\underline{x} \rightarrow \underline{x} = G'^{-1}\underline{y}'$$

where $\underline{x}$ is the source data and $\underline{y}'$ is a subset of k components of $\underline{y}$ available at the receiver. Matrix G' is the subset of rows from G corresponding to the components of $\underline{y}'$.

It is useful to solve the problem in two steps: first G' is inverted, then $\underline{x} = G'^{-1}\underline{y}'$ is computed. This is because the cost of matrix inversion can be amortized over all the elements which are contained in a packet, becoming negligible in many cases.

The inversion of G' can be done with the usual techniques, by replacing division with multiplication by the inverse field element. The cost of inversion is $O(kl^2)$, where $l \leq \min(k, n-k)$ is the number of data blocks which must be recovered (very small constants are involved in our use of the $O()$ notation).

Reconstructing the l missing data blocks has a total cost of $O(lk)$ operations. Provided sufficient resources, it is not impossible to reconstruct the missing data in constant time, although this would be pointless since just receiving the data requires $O(k)$ time. Many implementations of error correcting codes use dedicated hardware (either hardwired, or in the form of a dedicated processor) to perform data reconstruction with the required speed.

3 An erasure code based on Vandermonde matrices

A simple yet effective way to build the generator matrix, G, consists in using coefficients of the form

$$g_{ij} = x_i^{j-1}$$

where the x_i's are elements of $GF(p^r)$. Such matrices are commonly known as Vandermonde matrices, and their determinant is

$$\prod_{i,j=1\ldots k, i<j} (x_j - x_i)$$

If all x_i's are different, the matrix has a non-null determinant and it is invertible. Provided $q > k$ and all $x_i \neq 0$, up to $q-1$ rows can be constructed, which satisfy the properties required for G. Such matrices can be extended with the identity matrix I_k to obtain a suitable generator for a systematic code.

Note that there are some special cases of the above code which are of trivial implementation. As an example, an $(n, 1)$ code simply requires the same data to be retransmitted multiple times, hence there is no overhead involved in the encoding. Another simple case is that of a systematic $(k+1, k)$ code, where the only redundant block is simply the sum (as defined in $GF(p^r)$) of the k source data blocks, i.e. a simple XOR in case $p = 2$. Unfortunately, an $(n, 1)$ code has a low rate and is relatively inefficient compared to codes with higher values of k. Conversely, a $(k+1, k)$ code is only useful for small amount of losses. So, in many cases there is a real need for codes with $k > 1$ and $n - k > 1$.

We have written a portable C implementation of the above code [16] to determine its performance when used within network protocols. Our code supports $p = 2$, any r in the range $2 \ldots 16$, and arbitrary packet sizes. The maximum efficiency can be achieved using $r = 8$, since this allows most operations to be executed using table lookups. The generator matrix has the form indicated above, with $x_i = \alpha^{i-1}$. We can build up to $2^n - 1$ rows in this way, which makes it possible to construct codes up to $n = 2(2^r - 1), k = 2^r - 1$. In our experiments we have used a packet size of 1024 bytes.

3.1 Performance

Using a systematic code, the encoder takes groups of k source data blocks to produce $n-k$ redundant blocks. This means that every source data block is used $n-k$ times, and we can expect the encoding time to be a linear function of $n-k$. It is probably more practical to measure the time to produce a single data block, which depends on the single parameter k. It is easy to derive that this time is (for sufficiently large packets) linearly dependent on k, hence we can approximate it as

$$\text{encoding time} = \frac{k}{c_e}$$

where the constant c_e depends on the speed of the system. The above relation only tells us how fast we can build redundant packets. If we use a systematic code, sending k blocks of source data requires the actual computation of $n-k$ redundant blocks. Thus, the actual encoding speed becomes

$$\text{encoding speed} = \frac{c_e}{n-k}$$

Note that the maximum loss rate that we can sustain is $\frac{n-k}{n}$, which means that, for a given maximum loss rate, the encoding speed also decreases with n.

Decoding costs depend on $l \leq \min(k, n-k)$, the actual number of missing source blocks. Although matrix inversion has a cost $O(kl^2)$, this cost is amortized over the size s of a packet; we have found that, for reasonably sized packets (say above 256 bytes), and k up to 32, the cost of matrix inversion becomes negligible compared to the cost of packet reconstruction, which is $O(lk)$. Also for the reconstruction process it is more practical to measure the overall cost per reconstructed block, which is similar to the encoding cost. Then, the decoding speed can be written as

$$\text{decoding speed} = \frac{c_d}{l}$$

with the constant c_d slightly smaller than c_e because of some additional overheads (including the already mentioned matrix inversion).

The accuracy of the above approximations has been tested on our implementation using a packet size of 1024 bytes, and different values of k and $l = n-k$, as shown in Table 1 (more detailed performance data can be found in [17]). Running times have been determined using a Pentium 133 running FreeBSD, using our code compiled with `gcc -O2` and no special optimizations.

These experimental results show that the approximation is sufficiently accurate. Also, the values of c_e and c_d are sufficiently high to allow these codes to be used in a wide range of applications, depending on the actual values of k and $l = n-k$. The reader will notice that, for a given k, larger values of l (which we have set equal to $n-k$) yield slightly better performance both in encoding and decoding. On the encoder side this is exclusively due to the effect of caching: since the same source data are used several times to compute multiple redundant blocks, successive computations find the operands already in cache hence running slightly faster. For the decoder, this derives from the amortization of matrix inversion costs over a larger number

	Encoding			Decoding	
k	time/pkt	c_e	l	time/pkt	c_d
	μs	MB/s		μs	MB/s
8	840	9.53	1	1230	6.50
8	773	10.34	7	871	9.19
16	1553	10.30	2	1996	8.02
16	1500	10.69	14	1754	9.12
32	3012	10.62	4	3623	8.83
32	2967	10.78	28	3533	9.06

Table 1: Encoding/decoding times for different values of k and $n - k$ on a Pentium 133 running FreeBSD

of reconstructed blocks[2].

Note that in many cases data exchanged over a network connection are already subject to a small number of copies (e.g. from kernel to user space) and accesses to compute checksums. Thus, part of the overhead for reconstructing missing data might be amortized by using integrated layer processing techniques [3].

3.2 Discussion

The above results show that a software implementation of erasure codes is computationally expensive, but on today's machines they can be safely afforded with little overhead for low-to-medium speed applications, up to the 100 KB/s range. This covers a wide range of real-time applications including network whiteboards and audio/video conferencing tools, and can even be used to support browsing-type applications. More bandwidth-intensive applications can still make good use of software FEC techniques, with a careful tuning of operating parameters (specifically, $n - k$ in our discussion) or provided sufficient processing power is available. The current trend of increasing processing speeds, and the availability of Symmetric MultiProcessor (SMP) desktop computers suggest that, as time goes by, there will likely be plenty of processing power to support these computations (we have measured values for c_d and c_e in the 30MB/s range on faster machines based on PentiumPRO 200 and UltraSparc processors). Finally, note that in many cases both encoding and decoding can be done offline, so many non-real-time application can use this feature and apply FEC techniques while communicating at much higher speeds than their encoding/decoding ability.

[2] and a small overhead existing in our implementation for non reconstructed blocks which are still copied in the reconstruction process

4 Applications

Depending on the application, ARQ and FEC can be used separately or together, and in the latter case either on different layers or in a combined fashion. In general, there is a tradeoff between the improved reliability of FEC-based protocols and their higher computational costs, and this tradeoff often dictates the choice.

It is beyond the scope of this paper to make an in-depth analysis of the relative advantages of FEC, ARQ or combinations thereof. Such studies are present in some papers in the literature (see, for example, [7, 12, 21]). In this section we limit our interest to computer networks, and present a partial list of applications which could benefit from the use of an encoding technique such as the one described in this paper. The bandwidth, reliability and congestion control requirements of these applications vary widely.

Losses in computer networks mainly depend on congestion, and congestion is the network analogue of noise (or interference) in telecommunications systems. Hence, FEC techniques based on a redundant encoding give us similar types of advantages, namely increased resilience to noise and interference. Depending on the amount of redundancy, the residual packet loss rate can be made arbitrarily small, to the point that reliable transfers can be achieved without the need for a feedback channel. Or, one might just be interested in a reduction of the residual loss rate, so that performance is generally improved but feedback from the receiver is still needed.

4.1 Unicast applications

In unicast applications, reducing the amount of feedback necessary for reliable delivery is generally useful to overcome the high delays incurred with ARQ techniques in the presence of long delay paths. Also, these techniques can be used in the presence of asymmetrical communication links. Two examples are the following:

- **forward error recovery on long delay paths**. TCP communications over long fat pipes suffer badly from random packet losses because of the time needed to get feedback from the receiver. Selective acknowledgements [13] can help improve the situation but only after the transmit window has opened wide enough, which is generally not true during connection startup and/or after an even short sequence of lost packets. To overcome this problem it might be useful to allocate (possibly adaptively, depending on the actual loss rate) a small fraction of the bandwidth to send redundant packets. The sender could compute a small number (1-2) of redundant packets on every group of k packets, and send these packets at the end of the group. In case of a single or double packet loss the receiver could defer the transmission of the dup ack until the expiration of a (possibly fast) timeout[3]. If, by that time, the group is complete and some of the redundant packets are available, then the missing one(s) can be recovered without the need for an explicit retransmission (this this would be equivalent to a fast retransmit). Otherwise, the usual congestion avoidance techniques can be adopted. A variant of RFC1323 timestamps[5]

[3] alternatively, the sender could delay retransmissions in the hope that the lost packet can be recovered using the redundant packets.

can be used to assign sequence numbers to packets thus allowing the receiver to determine the identity of received packets and perform the reconstruction process (TCP sequence numbers are not adequate for the purpose).

- **power saving in communication with mobile equipment** Mobile devices usually adopt wireless communication and have a limited power budget. This results in the need to reduce the number of transmissions. A redundant encoding of data can practically remove the need for acknowledgements while still allowing for reliable communications. As an example, a mobile browser can limit its transmissions to requests only, while incoming responses need not to be explicitly ACKed (such as it is done currently with HTTP over TCP) unless severe losses occur.

4.2 Multicast applications

The main field of application of redundant encoding is probably in multicast applications. Here, multiple receivers can experience losses on different packets, and insuring reliability via individual repairs might become extremely expensive. A second advantage derives from the aforementioned reduced need for handling a feedback channel from receivers. Reducing the amount of feedback is an extremely useful feature since it allows protocols to scale well to large numbers of receivers.

Applications not depending on a reliable delivery can still benefit from a redundant encoding, because an improved reliability in the transmission allows for more aggressive coding techniques (e.g. compression) which in turn might result in a more effective usage of the available bandwidth.

A list of multicast applications which would benefit from the use of a redundant encoding follows.

- **videoconferencing tools**. A redundant encoding with small values of k and $n - k$ can provide an effective protection against losses in videoconferencing applications. By reducing the effective loss rate one can even use a more efficient encoding technique (e.g. fewer “I” frames in MPEG video) which provide a further reduction in the bandwidth. The PET [9] group at Berkeley has done something similar for MPEG video.

- **reliable multicast for groupware**. A redundant encoding can be used to greatly reduce the need for retransmissions (“repairs”) in applications needing a reliable multicast. One such example is given by the “network whiteboard” type of applications, where reliable transfer is needed for objects such as Postscript files or compound drawings.

- **one-to-many file transfer on LANs**. Classrooms using workstations often use this pattern of access to files, either in the booting process (all nodes download the kernel or startup files from a server) or during classes (where students download almost simultaneously the same documents or applications from a centralized server). While these problems can be partly overcome by preloading the software, centralized management is much more convenient and the use of a multicast-FTP type of application can make the system much more scalable.

- **one-to-many file transfer on Wide Area Networks**. There are several examples of such an application. Some popular Web servers are likely to have many simultaneous transfers of the same, large, piece of information (e.g. popular software packages). The same applies to, say, a newspaper which is distributed electronically over the network, or video-on-demand type of applications. Unlike local area multicast-FTP, receivers connect to the server at different times, and have different bandwidths and loss rates, and significant congestion control issues exist [8]. By using the encoding presented here, source data can be encoded and transmitted with a very large redundancy ($n >> k$). Using such a technique, a receiver basically needs only to collect a sufficient number (k) of packets per block to reconstruct the original file. The RMDP protocol [18] has been designed and implemented by the author using the above technique.

5 Acknowledgements

The author wishes to thank Phil Karn for discussions which led to the development of the code described in this paper, and an anonymous referee for comments on an early version of this paper.

References

[1] R.E.Blahut, "Theory and Practice of Error Control Codes" Addison Wesley, MA, 1984

[2] R.E. Blahut, "Fast Algorithms for Digital Signal Processing", Addison Wesley, 1987

[3] D.Clark, D.Tennenhouse, "Architectural Considerations for a New Generation of Protocols", ACM SIGCOMM'90, Sept. 1990, Philadelphia, pp.200-208.

[4] C.Huitema, "The Case for packet level FEC", Proc. 5th Workshop on Protocols for High Speed Networks, pp.109-120, Sophia Antipolis, France, Oct.1996.

[5] V. Jacobson, R. Braden, D. Borman, "RFC1323: TCP Extensions for High Performance", May 1992

[6] S.Lang, "Algebra", Addison-Wesley, 1984

[7] H.Liu, M. El Zarki, "Delay Bounded Type-II Hybrid ARQ for Video Transmission over Wireless Networks", Proc. Conference on Information Sciences and Systems, Princeton, NJ, March 1996

[8] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", ACM SIGCOMM'96, August 1996, Stanford, CA, pp.1-14.
Available as `ftp://ftp.ee.lbl.gov/papers/mccanne-sigcomm96.ps.gz`

[9] A.Albanese, J.Bloemer, J.Edmonds, M.Luby, M.Sudan, "Priority Encoding Transmission", 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Science Press, 1994.

[10] A. McAuley, “Reliable Broadband Communication Using A Burst Erasure Correcting Code”, Proc. SIGCOMM ’90.

[11] S.Lin, D.J.Costello, “Error Control Coding: Fundamentals and Applications”, Prentice Hall, 1983.

[12] S.Lin, D.J.Costello, M.Miller, “Automatic-repeat-request error-control schemes”, IEEE Comm. Magazine, v.22,n.12, pp.5-17, Dec.1984

[13] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “RFC2018: TCP Selective Acknowledgement Option”, Oct.1996.

[14] Jorg Nonnenmacher, E.W.Biersack, “Reliable Multicast: Where to use Forward Error Correction”, Proc. 5th Workshop on Protocols for High Speed Networks, pp.134-148, Sophia Antipolis, France, Oct.1996.
Available as `http://www.eurecom.fr/~nonnen/mypages/FECgain.ps.gz`

[15] V. Pless, “Introduction to Error-Correcting Codes”, 2nd ed., Wiley, 1989.

[16] L.Rizzo, Sources for an erasure code based on Vandermonde matrices.
Available at `http://www.iet.unipi.it/~luigi/vdm.tgz`

[17] L.Rizzo, “On the feasibility of software FEC”, DEIT Technical Report LR-970131. Available as `http://www.iet.unipi.it/~luigi/softfec.ps`

[18] L.Rizzo, L.Vicisano, “A Reliable Multicast data Distribution Protocol based on software FEC techniques”, DEIT Technical Report LR-970116. Available as `http://www.iet.unipi.it/~luigi/rmdp.ps`

[19] N.Shacham, P.McKenney, “Packet recovery in high-speed networks using coding and buffer management”, Proc. IEEE Infocom’90, San Francisco, CA, pp.124-131, May 1990.

[20] J.H. van Lint, “Introduction to Coding Theory”, 2nd ed., Springer-Verlag, 1992.

[21] Y. Wang, S.Lin, “A modified selective-repeat type-II hybrid ARQ system and its performance analysis”, IEEE Trans. Comm. v.COM-31, n.5, pp.593-608, May 1983