On the feasibility of software FEC

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

January 31, 1997– A copy of this paper is available as
`http://www.iet.unipi.it/~luigi/softfec.ps`

Abstract

Forward Error Correction (FEC) is seldom used in computer networks, because of perplexity in doing the necessary encoding/decoding in software. We believe this diffidence to originate from the fact that error control codes (which FEC is usually based upon) are generally designed and presented stressing their ability to correct *errors* rather than *erasures* (i.e. missing data in known locations). Handling erasures is a much simpler computational problem, and can be approached and solved with much simpler techniques.

This paper has three main objectives. First, to put software FEC in the right perspective by presenting the principle of operation of erasure codes and showing that software FEC is not exceedingly expensive. Second, to provide the research community with a high performance, C implementation of erasure codes capable to run at speeds in excess of 100 Mbit/s. Third, to present a number of applications (some of which have been developed, by the author or other researchers) where software FEC can, despite its overhead, be used to actually improve performance. These application cover slow, wireless networks and fast wired networks, as well as unicast and multicast communication protocols. Our conclusion is that software FEC is feasible and useful in a number of applications, and does not require supercomputing power to be implemented.

Keywords: FEC, erasure codes, reliable multicast, wireless networks, PDA's

1 Introduction

Reliability in computer communications is achieved either by using reliable transport protocols such as TCP, or by implementing the required features directly in the application. In both cases, the unavoidable packet losses are recovered using ARQ or FEC techniques, or combinations thereof.

ARQ is generally used in unicast protocols. Since no manipulations of the data stream are required, and only missing packets are retransmitted, ARQ is extremely inexpensive. In presence of losses, when the bandwidth-delay product approaches the sender's window, ARQ might result in a reduced throughput; also, data recovery requires at least an additional RTT, which can make this technique unattractive when high latencies cannot be tolerated. Finally, the effectiveness of ARQ decreases in multicast communication protocols when the number of (groups of) clients with uncorrelated losses grows.

FEC prevents losses by transmitting redundant information, which allow the reconstruction of some amount of missing data at the receiver without the need for retransmissions. In addition to a faster recovery of missing packets, FEC can help in simplifying protocols (because of large improvements in the packet loss rate), and might even render a feedback channel unnecessary.

FEC is very attractive for multicast applications since different loss patterns can be recovered using the same set of transmitted data. On the negative side, though, FEC is computationally expensive since the encoding/decoding requires some processing of the data stream.

As a matter of fact, FEC is less expensive than what is commonly thought, and one of the objectives of this paper is to show that FEC can be implemented in software without an excessive overhead. FEC is generally based on the use of error control codes (ECC), which have been studied for a long time and are widely used in many fields of information processing, particularly in telecommunication systems. In that area, the main requirement for such codes is the ability to recover from *errors*, i.e. wrong bit patterns in unknown positions. In the context of computer communications, error detection is generally provided by the lower protocol layers, and corrupt packets are discarded. The upper protocol layers thus only have to deal with *erasures*, i.e. missing data in a stream. Erasures are easier to deal with than errors, since the exact position of missing data is known.

In the first part of this paper we provide a simplified description of the principles of operation of erasure codes. We then present an erasure code which is easy to understand, flexible and highly efficient to implement even on inexpensive architectures. The complexity of our code is evaluated theoretically, and actual performance data is determined by running our publicly available C implementation [19] on a number of architectures. These data show that FEC can be profitably used on a wide range of applications and on a wide range of systems, from Portable Digital Assistants (PDA's) to high-end workstations. Basing on these data, the last part of the paper presents a number of applications where software FEC can be profitably used. These application cover both wireless and wired networks, and unicast and multicast communication protocols.

2 An introduction to erasure codes

An (n, k) erasure code encodes k *source* data items into $n > k$ *encoded* data items. The encoding is such that any subset of k encoded items suffices to reconstruct the source data. Figure 1 gives a graphical representation of the encoding and decoding process. Typically, all the n data items are transmitted, so that the receiver can recover from up to $n - k$ losses in a group of n. k/n is called the *rate* of the code, i.e. the fraction of transmitted data which represent the source information.

Erasure codes are generally used to reduce the loss rate on a noisy channel. If p is the loss rate for a single data item, using an (n, k) code the residual loss rate decreases roughly as p^{n-k}. Depending on the actual needs, this number can be made arbitrarily small.

In some cases, producing an erasure code is trivial: as an example, an $(n, 1)$ code simply consists in the repetition of the source data. If the loss rate is p, after encoding the residual loss rate becomes p^n, i.e. it decays exponentially with p. This comes of course at a price in the bandwidth, since the rate of this code is $1/n$, very low. Another trivial code to build is a $(k+1, k)$ code. In this case, a single redundant item is built using k source data items. The easiest way to build the redundant packet consists in XOR-ing all the source data. At the receiver, recovery of any single missing data item is simply done by XOR-ing all the received data. This time, a non redundant encoding would allow the successful transfer of k packets with probability $(1 - p)^k$; for $kp \ll 1$, as it is generally the case, the failure rate can be approximated by kp. The use of a redundant encoding brings the failure rate to approximately $(k + 1)kp^2/2$, i.e. significantly smaller than the original value; this time the rate of the code is $k/(k + 1)$, insuring a good use of the bandwidth.

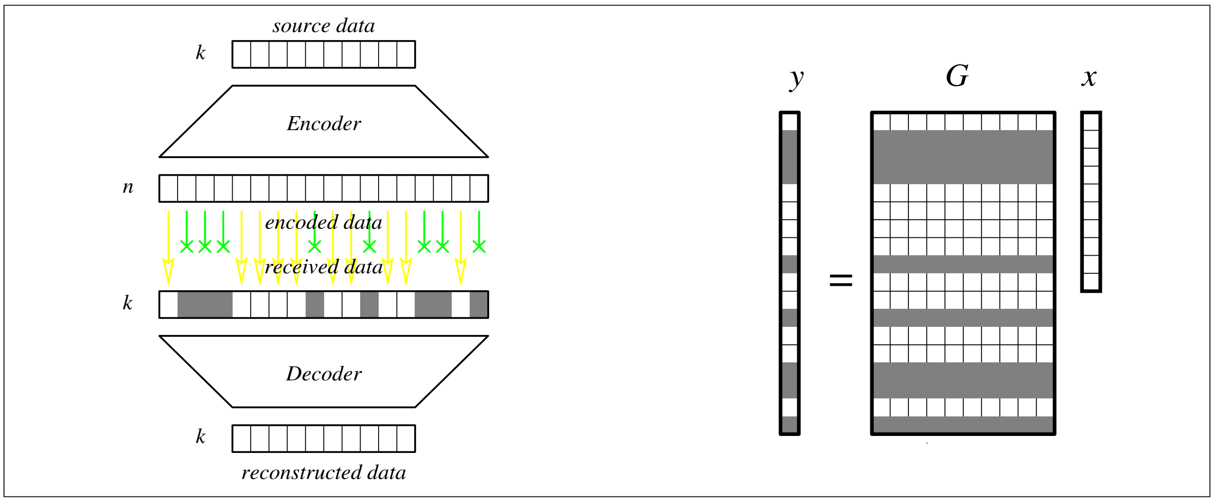

Figure 1: A representation of the encoding/decoding process, and its equivalent in matrix representation. Grey areas represent missing blocks (or components of y). G' is made of the white rows of G.

The two erasure codes presented this far have been derived without basing on any specific theory: they simply appear to work, and can be easily proved to behave as expected. Codes with generic n and k cannot be designed so easily, and a more systematical approach is needed.

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 $\underline{x} = x_0 \ldots x_{k-1}$ be the source data, G an $n \times k$ matrix, then an (n, k) linear erasure code can be represented by

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

where G is such that any submatrix G' made of k rows from G is invertible. If the encoded data include a verbatim copy of the source data items, 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.

Assuming that $\underline{y}'$, made of k components of $\underline{y}$, is available at the receiver, source data can be reconstructed by using the k equations corresponding to the known components of $\underline{y}$. If we call G' the $k \times k$ matrix representing these equations (see Figure 1), reconstructing the source data is possible by solving the linear system

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

Strictly speaking, the reconstruction process needs some additional information – namely, the identity of the received components of $\underline{y}$ – 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 typical packet sizes.

There is however another source of overhead which cannot be neglected, and this is the precision used for computations. Using ordinary arithmetic, if x_i's are represented with b bits, and g_{ij}'s with 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.

2.1 Finite fields

The expansion of encoded data can be avoided by doing computation in a finite field. Finite fields have $q = p^n$ elements, where p is a prime number, and a sum and product operator (and their inverse) to operate on field elements with analogous properties to the ordinary sum and product. Most of the properties of linear algebra apply to finite fields as well.

Fields with $q = p$ elements, with p prime, are called *prime fields* or $GF(p)$. Operating in a prime field is relatively simple, since field elements can be thought of as integers modulo p, and sum and product are just the ordinary sum and product, with results computed 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 cannot be represented with an integer number of bits. 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. Secondly, an expensive modulo operation is required to perform sums and, especially, multiplications. Both problems, though, can be minimized if $p = 2^m + 1$i, since in this case the modulo operation can be done cheaply using shifts and sums.

Fields with $q = p^n$ elements, with p prime and $n > 1$, are called *extension fields* or $GF(p^n)$. Sum and product in extension fields are more complex than in prime fields, since they *cannot* be done by taking results modulo q. Rather, field elements can be considered as polynomials of degree $n - 1$ whose coefficients are integers modulo p. The sum operation is just the sum between coefficients, modulo p; the product is the product between polynomials, computed modulo a special polynomial of degree n, 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 $GF(2^n)$ require exactly n 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.

2.1.1 Multiplication and division

Multiplication and division appear much more complex, at least at first sight. However, the following properties allow considerable simplifications in their implementation.

The number of different values produced by powers of a field element x is called the *order* of x. An interesting property of prime or extension fields is that there exist at least one element (usually denoted by α) of order $q - 1$, whose powers generate all non-zero elements of the field. As an example, 2 has order 4 in $GF(5)$, and its powers (starting from 2^0) are $1, 2, 4, 3, 1, \ldots$. Powers of α repeat with a period of length $q - 1$, hence $\alpha^{q-1} = \alpha^0 = 1$.

The existence of an element of order $q-1$ 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:

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

If the number of field elements not too large, tables can be built off line to provide the "logarithm", the "exponential" and the inverse of each field element. In some cases, it can be convenient to provide a table for multiplications as well. Using the above technique, multiplication and division require just one table lookup if q is small and a table of size q^2 can be afforded, or three table lookups and one sum otherwise.

3 A simple erasure code

Having solved the problem of operand expansion, the only difficulty in building an erasure code remains to find a suitable encoding matrix G. One way to build G is to interpret the k source data items as the coefficients of a polynomial of degree $k-1$. It is well known that such a polynomial is completely specified by giving its value at k different points. This property holds even if computations are done in a finite field, provided the field has at least k distinct elements. As a consequence, if we compute the value of the polynomial at n different points, any subset of k of these values makes it possible to reconstruct the polynomial. The matrix G relating the coefficients of the polynomial, and its values at n different points p_i, has coefficients

$$G = \{g_{ij}\} = \left\{p_i^{j-1}\right\}$$

Such matrices are commonly known as Vandermonde matrices. Provided $q > k$, up to q rows can be constructed (one for each point in $GF(q)$) which satisfy the properties required to an encoding matrix. If all $p_i \neq 0$, G can be extended with the identity matrix I_k to obtain a suitable generator for a systematic code. The $(n,1)$ and $(k+1,k)$ codes we have presented earlier can be considered a special case of our code. The former represents a degree-0 polynomial (i.e. a constant) which obviously has the same value at all points (thus G becomes an $n \times 1$ matrix with unity coefficients). The latter represents a degree $k-1$ polynomial evaluated at $p = 1$ (this time G is a $1 \times k$ matrix with unity coefficients), yielding as a result the sum of all coefficients; in $GF(2^n)$, this corresponds to XOR-ing all k source data.

3.1 Performance

We have written a portable, publicly available C implementation of the above code [19] to be used in the implementation of network protocols. Our code operates in $GF(2^b)$, with $b = 2 \ldots 16$. The generator matrix has the form indicated above, with $p_i = \alpha^{i-1}$, and is extended with I_k to produce a systematic code. We can construct systematic codes up to $\left(2(2^b-1), 2^b-1\right)$ this way.

A comment about data sizes is necessary at this point. In order for the encoding technique to work, data items must have a sufficient size to represent field elements, i.e. b bits, with practical values of b in the range $4 \ldots 16$ (at least for a software implementation). In computer networks, data is exchanged in packets which are generally much larger than 16 bits. The encoding/decoding algorithm can be applied to whole packets by considering them as made of s elements of b bits each. Since packets are dealt with by the network as a whole, there is actually an advantage in working on whole packets: in fact, during decoding, the same matrix G'^{-1} can be used for all elements of a packets, with significant savings in the computations.

With this in mind, we can start the analysis of the complexity of the encoding/decoding algorithms. The encoder uses k source data packets to produce each encoded packet. As a consequence the time t_e to produce a single encoded packet of size s can be approximated as

$$t_e = \frac{ks}{c_e}$$

where the constant c_e is an indication of 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 overhead is amortized over the whole block, and the total encoding time for a block of k packets becomes $T_e = (n-k)t_e$; thus, the encoding speed becomes

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

Note that the maximum loss rate that we can sustain is $(n-k)/n$. Hence, encoding speed and sustainable loss rate are not directly related; for a given maximum loss rate, the encoding speed decreases with n. This is expected: for a given loss rate, a large block implies a more powerful code, since there is more freedom on the location of erasures.

Decoding costs are slightly harder to determine: in fact, the decoding process must first perform a matrix inversion to determine G'^{-1}, then perform a matrix multiplication to recover the l missing source packets ($l \leq \min(k, n-k)$). Matrix inversion requires $O(kl^2)$ operations per block of k packets, whereas the subsequent matrix multiplication requires $O(k)$ operations for each reconstructed data item, or a total of $O(kls)$ operations per block of k packets. This shows clearly that, for reasonably sized packets, the cost of matrix inversion becomes negligible compared to the cost of packet reconstruction (see also Table 1, where the various costs are detailed). Under these assumptions, the cost of the decoding process is similar to encoding, and the decoding speed can be written as

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

with the constant c_d slightly smaller than c_e to account for the additional overheads (including the already mentioned matrix inversion).

3.2 Experimental results

The accuracy of the above approximations has been tested with our implementation working in $GF(2^8)$, using a packet size $s = 1024$ bytes, and different values of k and l, as shown in Table 1. Running times have been determined using a Pentium133. These results show that the approximation is sufficiently correct: c_e and c_d remain approximately constant[1] over a wide range of values for k and l.

Smaller packet sizes make the overall performance worse, but not dramatically so. Figure 2 plots the dependency of decoding speed on packet sizes, normalized to $s = 1024$. The relative weight of matrix inversion grows as s gets smaller, especially for large k and l. However, it is interesting to note that even for very small packets (e.g. the 48 bytes of payload used in ATM cells) a reasonable level of performance can still be achieved.

Table 2 shows the value of c_d on different architectures, using our portable C implementation. The table voluntarily includes a wide range of systems, from high end workstations to old machines and small portable systems, since FEC can be used in very different contexts and with very different speed requirements. The reader is warned to take these results with some caution, since the absolute values are extremely sensitive to the quality of the compiler, the cache and memory bandwidth, the CPU type. We have compiled the code with `gcc -O3` where available, but without using more aggressive optimizations, or machine-specific optimizations in

[1] c_d appears smaller for $l = 1$, but this is only an artifact of our implementation, which does a `bcopy()` of non-reconstructed packets. Removing this overhead actually makes c_d slightly higher for small values of l.

k	l	Encoding time/pkt t_e (μs)	c_e MB/s	Matrix inversion time (μs)	Total decoding time μs	c_d MB/s
32	1	2500	12.800	195	3302	9.199
28	1	2016	13.891	142	3020	9.494
24	1	1751	13.707	106	2537	9.687
20	1	1433	13.960	82	2084	9.827
16	1	1213	13.181	59	1716	9.547
12	1	868	13.822	38	1271	9.668
8	1	580	13.791	14	436	9.394
32	28	2194	14.583	6314	79014	11.612
28	24	1922	14.569	4133	58866	11.690
24	21	1643	14.602	2818	44030	11.721
20	17	1371	14.586	1594	29371	11.811
16	14	1103	14.512	892	19421	11.811
12	10	827	14.508	399	10332	11.893
8	7	557	14.372	146	4870	11.775
4	3	289	13.838	26	1084	11.336

Table 1: Encoding/decoding times and speeds for different values of k and l, measured on a Pentium133.

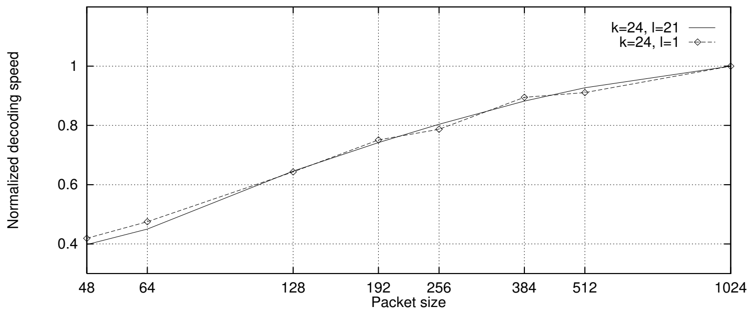

Figure 2: c_d for different packet sizes, normalized to the value for $s = 1024$

System	CPU/Speed	c_d (MB/s)
Sun Ultra1	UltraSPARC 167 MHz	21.289
DEC	Alpha 255 MHz	12.577
PC/FreeBSD	Pentium 133 MHz	11.893
HP715/100	PA7000 100 MHz	10.308
SPARCstation-20	Sparc TMS390Z55 60 MHz	8.091
SPARCstation-10	Sparc TMS390Z55 40 MHz	5.391
SGI Indy	R4000	5.186
SPARCstation-5	SPARC MB86904 70 MHz	3.947
PC/FreeBSD	i486/DX2 66 MHz	3.338
Sun IPX		2.124
DECstation 2100	R2000 8 MHz	0.612
PC/FreeBSD	i386 25 MHz	0.372

Table 2: Operating speed of the encoding/decoding process on various architectures

the source code. Most of the computation time is spent in doing the multiply-and-add operation typical of matrix multiplications. Due to lack of native CPU support for working in finite fields, this operation involves two memory reads, one XOR and one memory write on byte operands. As a consequence, we have good reasons to believe that handcrafted code which makes good use of architectural features of the system (such as large number of registers, cache line widths, etc.) might achieve large speedups on the values presented in Table 2. Ideally, c_d and c_e should be very close to $1/5 \ldots 1/4$ of the memory bandwidth of the system. The application of IFP principles [10] can obviously make this overhead even smaller.

Also, simple cases such as $(k+1, k)$ codes lead to further optimizations, by simply XOR-ing packets instead of doing an expensive matrix multiplication. In this case, the value of c_e and c_d can become very close to $1/2$ of the memory bandwidth of the system. Implementing a $(k+1, k)$ code has at most twice the cost of computing IP or TCP checksums.

The above results show that a software implementation of erasure codes is computationally expensive, but on todays' machines it can be safely afforded at speeds of 100 Mbit/s and higher (depending on the system and the actual encoding parameters of course). The current trend of progress in processing speeds, and the availability of Symmetric MultiProcessor (SMP) desktop computers suggest that, as time goes by, there will be plenty of processing power to support these computations. 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 use FEC when the communication speed is much higher than the encoding/decoding speed.

4 Applications

The range of applications which can take advantage of FEC techniques is extremely wide, and in this section we present a partial list of them. The bandwidth, reliability and congestion control requirements of these applications vary widely, as well as the communication patterns. However, the common requirement is for an improved behaviour in presence of packet losses, to whatever reason they depend upon. FEC gives an increased resilience to losses, and can be superior to ARQ when the number of packets to transmit is sufficiently large that the encoding overhead is not exceedingly larger than the expected loss rate. For a proper choice of parameters, FEC can make the residual packet loss rate so small that reliable transfers can be achieved without the need for a feedback channel. This is an extremely important feature for the development of scalable multicast protocols. Additionally, by avoiding the need of an explicit retransmission to recover missing data, the jitter in communication latency can be significantly reduced, making the transport better suited to the transfer of real-time data.

In unicast applications, reducing the amount of feedback necessary for reliable delivery is generally useful to overcome the high delays incurred with ARQ in presence of long delay paths. Also, FEC can be used in presence of asymmetrical communication links, e.g. when uplink communication is expensive and must be minimized.

Reliable multicast communication is the area where FEC can be most beneficial. In fact, ARQ scales poorly with the number of receivers because of two reasons: first, the sender might have to deal with an exceedingly number of ACKs (or NACKs) from the receivers (the "ACK implosion" phenomenon); second, as the number of receivers grows, so does the probability that losses at different receivers become uncorrelated, to the point that most if not all the packets need to be retransmitted at least once, with a large impact on performance. Recently, several papers [13, 18] have addressed this problem showing how a moderate amount of redundancy can drastically reduce the need for retransmissions. Despite this, almost all reliable multicast protocols that have been presented in the literature, even in recent years [12, 16, 22] are based on ARQ instead of FEC, often using very ingenious and complex techniques trying to overcome the intrinsic limitations of ARQ. Of course, by reducing the perceived loss rate, the number of ACKs/NACKs decreases correspondingly, making the techniques used to avoid ACK implosion [11] even more effective.

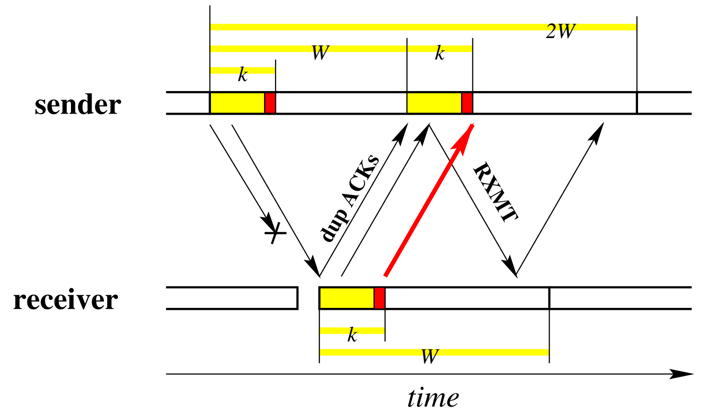

Figure 3: TCP windows with and without FEC. By using FEC, the addition of a small number of redundant packets (dark) every k source ones, makes loss recovery much faster (thick arrow) and reduces the window requirements in presence of recoverable losses from $2W$ to $W + k$ and from W to k at the sender and receiver, respectively.

4.1 TCP and FEC

Let $W = B \cdot RTT/s$ be the bandwidth-delay product of a TCP connection, measured in packets (of size s each). In order to keep a TCP connection streaming, in absence of losses the sender's window must be at least W. The window must be further doubled to $2W$ if we want to keep streaming in case of losses that can be recovered using the Fast Retransmit or Selective Acknowledgement mechanisms (see Figure 3), since the sender cannot dispose of the whole window until the ACK for the retransmitted packet has come back. On fast, long delay networks such windows can easily fall into the MB range, requiring an equivalent amount of storage at the sender and the receiver, and possibly overflowing queues at intermediate routers if the bandwidth is temporarily reduced because of competing traffic. A second, well known problem with retransmission-based recovery is the inadequacy in carrying real-time traffic, since losses require at least 1 RTT to be recovered.

FEC can be useful both to reduce the window size requirement for an equivalent level of performance, and to reduce the time needed to recover from losses. By introducing a small number of redundant packets every k packets, occasional losses can be recovered and, importantly, ACKed, without the additional RTT required for a retransmission (Figure 3, thick arrow). Thus, a sender's window of size $W + k$ with FEC insures the same level of performance of a window of size $2W$ without FEC. At the receiver, k packet buffers will suffice, with very high probability, to store out-of-sequence data. Using k in the range 20..50, and 1-2 redundant packets per block, the waste of bandwidth can be kept within an acceptable 5% (consider that IP and TCP headers typically consume 3-5% of the available bandwidth) while allowing considerable savings on the window size. The performance data in Table 2 shows that the encoding/decoding speed can be well beyond the typical bandwidths available on long distance paths because of the small number of redundant packets.

The above problem might be of interest only for a small number of users – e.g. those interested in maximizing the usage of an expensive satellite link. But there are other, much more common, situations in which FEC can be extremely effective. In today's congested Internet, a packet loss rate in the 10-30% range is often experienced. Such a loss rate makes the network almost unusable for interactive applications, but can still be tolerated for bulk data transfers (such as FTP, email, news, and, to some degree, Web traffic). Unfortunately, in such conditions, TCP tends to lose its ACK-based self-clock, causing frequent timeouts and lost throughput. Using FEC, this time with a larger amount of redundancy and larger block sizes, performance can be largely improved. In fact, because of the encoding, packets need not to be retransmitted; when dup ACKs are received, different redundant packets can be sent every time, so that there is no risk of sending duplicate information and dup ACKs can be used more effectively, with the additional benefit of keeping data flowing, thus further reducing the chance that a connection loses its self clock and has to resort to a timeout to restart. Finally, the relatively large number of redundant packets limits the encoding/decoding speed, but this is not a relevant issue since the available bandwidth is already much more limited.

It is important to note that, in both situations, the conventional ACK-based congestion control mechanisms are not affected by the use of FEC: dup ACKs are still generated when out-of-sequence data are received, and can be used by the sender to dynamically adjust the window size.

4.2 FEC in mobile communication

FEC is heavily used in data link protocols for wireless networks [3], to overcome the unreliability of the medium and provide an acceptable packet loss rate. But there are good reasons to adopt FEC at higher levels in the protocol stack. Mobile devices usually have a limited power budget; since transmission is expensive, it must be limited. One way to achieve such savings is to reduce or remove the need for acknowledgments, which can be done by using a sufficient amount of redundancy to make the residual loss rate negligible. As an example, a consider the case of a mobile device browsing the Web over a wireless connection. After the initial request to the server, data can be sent to the mobile unit with a highly (e.g. 20..50%) redundant encoding, and using large values of k, so that even in presence of severe losses (due to noise, congestion, or simply roaming) incoming data need not to be explicitly ACKed. The encoding can be done at the source, e.g. by negotiating the transport over a different protocol than TCP. This has been implemented by the author as a special case of the protocol described in Section 4.3. Alternatively, the base station connected to the wireless link can buffer data and acknowledges the transmitter in lieu of the mobile unit, implementing FEC on the wireless link only. A similar approach, using ARQ instead of FEC, has been used in [4]. In our case, FEC gives the additional advantage that the amount of feedback from the receiver can be greatly reduced thus allowing significant power savings, or operation over asymmetrical channels (e.g. where requests are sent over a phone line, whereas the downstream is supported by a broadcast channel). Also in this case the encoding/decoding overhead is not a limiting factor: the typical speeds for this type of communications are in the 100 Kbit/s range, so that large values of $n - k$ are affordable even with low-end PDA's.

4.3 Reliable multicast data distribution

There are many applications which require the reliable distribution of data to relatively large groups of users (Figure 4). A first example is given by computer labs: here, in many cases a relatively large (10..100) number of clients accesses the same data (e.g. a kernel during the booting process, application programs or slides during lectures) almost at the same time. In this case, a large bandwidth is available, and congestion control (an open research issue otherwise [15]) is not of concern since communication is local and there is only one server.

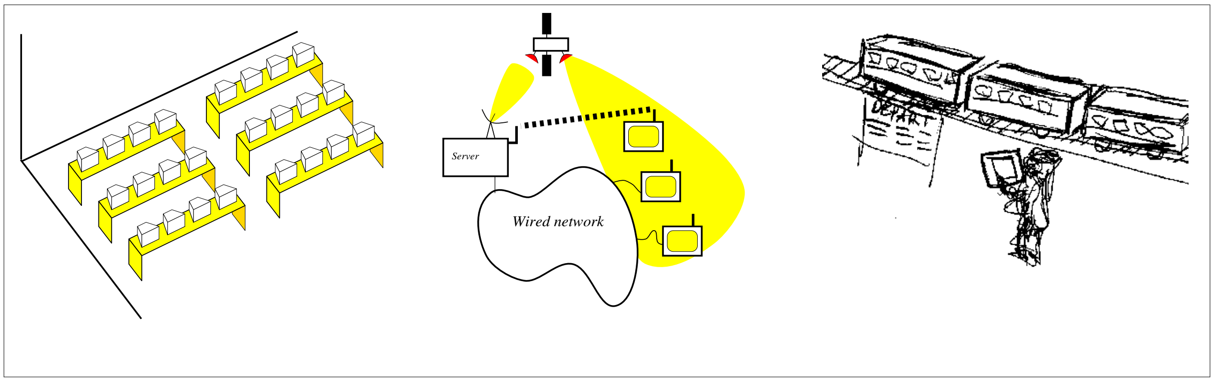

Figure 4: Some possible applications of reliable multicast: file access in local area networks (e.g. a computer lab); wide area data distribution, over wireless or asymmetrical networks; and information retrieval in small geographical areas (stations, airports) using wireless equipment

Similar features are present in the distribution of documents to large groups of users, which are not necessarily in close proximity. There are many examples of this class of applications: distribution of electronic newspapers or stock quotes over the network (or even over radio/TV broadcast channels); access to schedules, maps or other info in stations, airports or highways, using portable equipment; access to information in museums or other touristic places; broadcast disks [1]; and, finally, distribution of frequently accessed Web pages such as home pages in popular search engines or sites distributing software. In all these cases the set of simultaneous receivers is large (from tens to thousands), documents are large, and reliable delivery is required because of the very nature of the documents. The problem of congestion control can be solved by artificially limiting the data rate at the sender, similarly to what is done with videoconferencing tools.

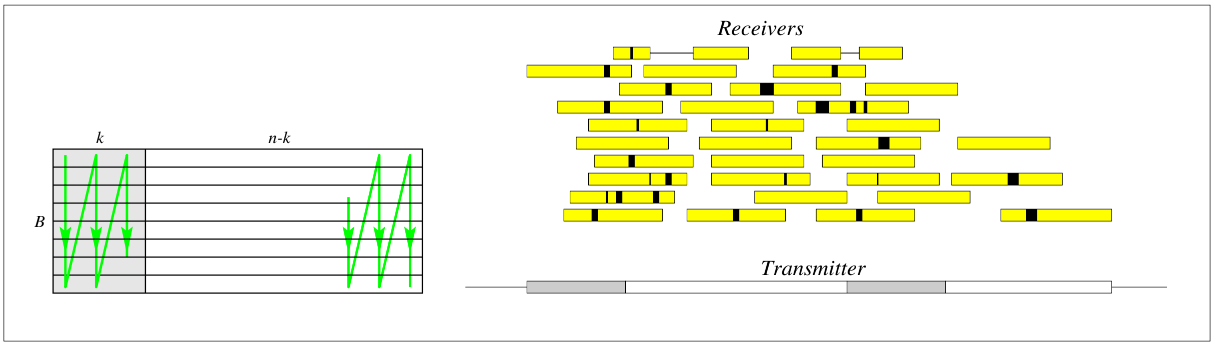

Figure 5: Left: RMDP sends data interleaved and encoded with high redundancy. Right: receivers can join the group at arbitrary times, experience independent losses (black rectangles) or even stop and resume reception (top row) arbitrarily. The protocol provides a graceful degradation of performance depending on the actual loss rate.

We have developed a Reliable Multicast data Distribution Protocol (RMDP) which is well suited to these types of application. The protocol, which is described in detail in [20], is based on the assumption that receivers join and leave the group continuously (see Figure 5) and the transmitter would then have to send the same data almost continuously. Data is split in blocks of k packets, which are encoded with extremely high redundancy ($n \gg k$) and transmitted using multicast and a proper interleaving among blocks. A receiver joining the group only needs to receive k different packets per block, but it is not important which ones. The condition $n \gg k$ insures that the same packet is not retransmitted for a long time, thus improving the receive efficiency.

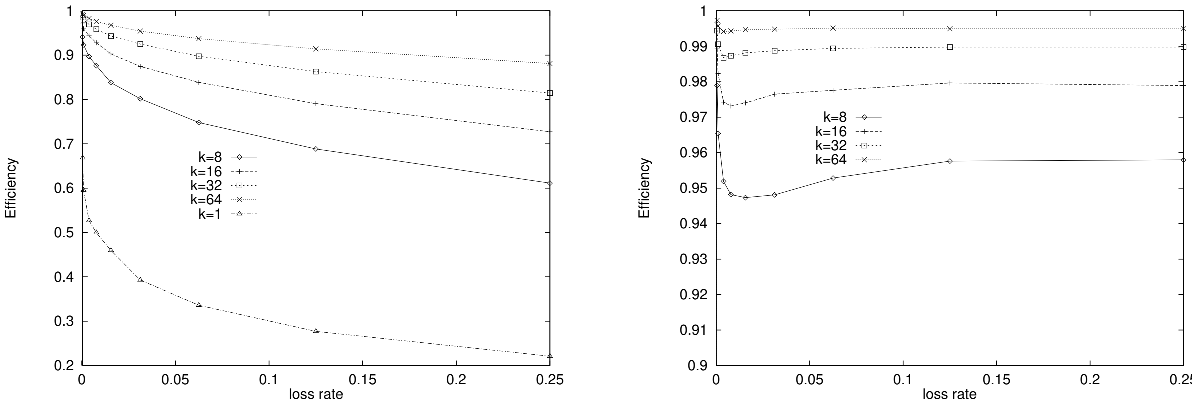

Figure 6: The efficiency of RMDP in case of random losses (left) and burst losses (right). The bottom line ($k = 1$) in the left plot does not represent a usable case, but it is shown to demonstrate how sensitive to losses is a protocol based on the simple repetition of source data, but without feedback from the receivers.

Figure 6 shows the performance of RMDP in presence of losses and no feedback whatsoever from the receivers, for a data size of 2048 packets. For a sufficient value of k (32 or more) the receiver efficiency (measured as the ratio between the minimum theoretical reception time and the actual reception time) is extremely high even under substantial random losses. The behaviour with burst losses is even better due to the interleaving used in the transmission.

RMDP supports groups of heterogeneous receivers with different speeds and loss patterns, and is suited to a number of different needs, from local, medium speed, file transfer, to wide area broadcast data distribution. Because of the nature of the protocol, feedback from the receiver is not required, although it can be used, if uplink communication is available and cheap, to tune the transmission rate to the current set of receivers. Receivers which are slower than the transmission rate, or behind lossy links, simply need to remain active for a longer time. Thanks to the use of FEC, RMDP is scalable to very large groups, unlike ARQ-based protocols which perform poorly with large or heterogeneous groups of receivers.

4.4 Unreliable multicast

In some cases, data need not to be distributed reliably, often because they have a short lifetime. This is the case for audio/video streams, where missing packets cause transient clicks or missing frames and can be tolerated without much trouble, provided they are not too frequent. In such cases, it is however advisable that the quality of the communication degrades gracefully with the increase of losses. Unfortunately, the data stream often carries different types of information with different loss tolerance. As an example, in MPEG video, I-frames are more important than B or P frames, since the former provide the reference for interpreting the latter.

In general, whenever a data stream is made of different parts with different importance, a "graceful degradation" of performance with respect to losses is not always built into the native encoding, and must be guaranteed by using a suitable, additional encoding. The PET project [2] at Berkeley has followed this approach to provide a loss-tolerant encoding of video streams. Different parts of the stream (e.g I, B and P frames) are encoded with different redundancy so that a minimum level of quality can be achieved also in presence of large losses, and the quality of the output increases as losses decrease.

5 Related work

In early computer networks, the use of FEC was ruled out because of the low speed of computers of the age. The advent of ATM networks forced FEC to be reconsidered [5, 6, 9], because single cell losses can compromise the reception of a whole, much larger, datagram. The speed typically used in ATM networks does not (yet[2]) leave room to software implementations, though. The use of FEC in a transport protocol has been proposed in [14]. Again, the target operating speed (1 Gbit/s) required a hardware implementation of the encoder/decoder, which the paper demonstrates to be feasible. The code proposed in [14] is essentially the same as the one used in this paper, and both are in fact a special case of the Reed-Solomon or BCH codes [7] which have been known and used for many years in the telecommunication field. The presentation of erasure codes given in this paper is different from the one which is generally found in coding theory textbooks, because we are only concerned about erasures. Also, we provide a slightly

[2]our software implementation can already run at speeds in excess of 100 Mbit/s on general-purpose processors, and a similar level of performance can be achieved with an implementation based on current DSPs. It is not unreasonable to expect that, in the near future, dedicated processors with a little bit of hardware support for finite field computations will be able to execute FEC algorithms at speeds in the Gbit/s range.

different, and much more detailed description of the recovery process than [14], together with a detailed analysis of the encoding/decoding costs.

A number of theoretical studies on the effectiveness of FEC in communication protocols has appeared in recent years [21, 13, 18, 9] dealing with general applications or specific protocols (e.g. IP over ATM), and both unicast and multicast applications. There is general consensus that FEC can be profitably used, although the issue of doing encoding/decoding in software, and the associated overhead, has been rarely mentioned.

Software FEC based on redundant encoding is used the already mentioned PET project [2]. PET uses a linear block code over $GF(2^{16}+1)$ and is reportedly able to do decoding in real time for compressed video traffic (presumably in the $0.1\ldots1$ Mbit/s range). A binary distribution of PET-enhanced `vic` is available, but no source code has been made publicly available. A different type of FEC – based on the simultaneous transmission of a low-quality replica of the data stream – has been used with success for audio communication in the MICE project [8]. Also, many packages have recently appeared for audio/video communication over the Internet, which presumably employ, to different degrees, FEC using proprietary and undocumented techniques.

Finally, it is worth mentioning that FEC is used, generally with a simple $(k+1,k)$ encoding, in RAID systems. In that context a high performance software implementation is feasible since small values for k are generally used, and the relative simplicity of encoding/decoding brings the constants c_e and c_d very close to the memory bandwidth of the system.

6 Conclusions and future work

We have given a simplified description of the principle of operation of erasure codes, which Forward Error Correction (FEC) techniques are generally based upon. A high performance, C implementation of an erasure code has also been presented, which is able to run at very high speed even on desktop PCs. Our implementation demonstrates that software FEC is feasible at reasonable costs, and it can be effectively used in a number of applications, to support reliable data transfer, or even to guarantee a graceful degradation of performance in presence of losses. These results stimulate future research in many directions. First of all, some of the applications described in Section 4 are being implemented and/or their performance evaluated. A more thorough analysis of protocols and applications is also necessary to determine other possible applications of FEC. A second area of research refers to the development of faster implementations of erasure codes, which are needed for very fast networks and small cells such as those used in ATM. It is relatively simple to show that the complexity of encoding and, in some cases, decoding, can be reduced by using techniques similar to those applied to compute Fast Fourier Transforms. But a significant improvement in performance can come from the hardware. The computations required for FEC are similar to those used in many audio/video processing techniques; we can expect significant speedups in the next years, pushed by multimedia applications. The addition, in future processors, of hardware support for finite field computations would provide significant speedups at very low costs.

References

[1] S. Acharya, M. Franklin, S. Zdonik, "Dissemination-Based Data Deliveri using Broadcast Disks", IEEE Personal Communications, Dec. 1995, pp.50-60 Broadcast disks...

[2] 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

[3] E. Ayanoglu, S. Paul, T. LaPorta, K. Sabnani, R. Gitlin, "AIRMAIL: A Link-layer Protocol for Wireless Networks", Wireless Networks, Baltzer Science Publishers, Feb. 1995

[4] H.Balakrishnan, S.Seshan, E.Amir, R.Katz, "Improving TCP/IP Performance over Wireless Networks", 1^{st} ACM Conf. on Mobile Computing and Networking (Mobicom), November 95

[5] Azer Bestavros and Gitae Kim, "TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks", To appear in Infocom'97, April 1997, Japan. The TR (July 1996) version of it is available from BU at `http://www.cs.bu.edu/techreports`

[6] Ernst Biersack, "Performance Evaluation of Forward Error Correction in ATM Networks", Comm. of ACM, pp.248-257, Aug.1992

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

[8] J-C. Bolot, A. Vega-Garcia, "The Case for FEC-Based Error Control for Packet Audio in the Internet", to appear in ACM Multimedia Systems.

[9] G.Carle, "Towards scaleable error control for reliable multipoint services in ATM networks", 12^{th} Int. Conf. on Computer Communication, ICCC'95, Seoul, Korea, August 1995

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

[11] S.Deering, "RFC 1112: Host extensions for IP Multicasting", Aug. 1989

[12] S.Floyd, V.Jacobson, C.Liu, S.McCanne, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM)", ACM SIGCOMM 95, available as `ftp://ftp.ee.lbl.gov/papers/srm_sigcomm.ps.Z`

[13] 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.

[14] A.J.McAuley, "Reliable Broadband Communication Using a Burst Erasure Correcting Code", *ACM SIGCOMM'90*, Sept. 1990 Philadelphia, pp.297-306.

[15] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", SIGCOMM'96, available as `ftp://ftp.ee.lbl.gov/papers/mccanne-sigcomm96.ps.gz`

[16] Lin, John C. and Paul, Sanjoy, "RMTP: A Reliable Multicast Transport Protocol", IEEE INFOCOM '96, March 1996, pp.1414-1424.
Available as `ftp://gwen.cs.purdue.edu/pub/lin/rmtp.ps.Z`

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

[18] 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`

[19] L.Rizzo, "Effective erasure codes for reliable computer communication protocols", DEIT Technical Report LR-970115, available as http://www.iet.unipi.it/~luigi/fec.ps Source code available as http://www.iet.unipi.it/~luigi/vdm.tgz

[20] 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

[21] 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

[22] R.Yavatkar, J.Griffioen, M.Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications", ACM Multimedia 95.
Available as http://hill.lut.ac.uk/DS-Archive/acm95.ps