The case for packet level FEC

February 5, 1996

Abstract

Packet level forward error correction can be implemented by transmitting M redundancy packets after each set of N regular packets, so that all packets can be reconstructed if at least N out of $N + M$ are received. The idea is usually dismissed because the gain is not worth the additionnal transmission overhead and the increased computation load, but further analysis shows that this dismissal may be questioned.

There are in fact at least three environment where the reduced error rate proves very valuable. When multicasting data toward large groups, even a small individual error rate per recipient may result in large retransmission rates for the whole group and the use of redundancy will result in dramatic efficiency gains. In the case of long transmission delays, the use of redundancy helps maintaining the delivery delays within acceptable limits, even in presence of errors. When the receivers do not have enough memory resources to implement sophisticated retransmission techniques, forward error correction can compensate the relative inefficiency of cheap algorithms of the go-back N family.

Packet level forward error correction is not very difficult to implement. The level of redundancy can easily be tuned as a function of the network's characteristics. The additional robustness obtained through packet level redundancy helps implementing feedback control algorithms. In short, this seldom used technology could easily improve the performance of current transmission control protocols.

1 Zero defect and networking

Although packet level FEC has been proposed by many researchers, the idea is usually dismissed because the gain is not worth the pain. Implementing complex packet redundancy code in software is deemed difficult. It is costier than a checksum, at least as expensive as a memory move, thus likely to nearly double the computational load of transport control protocols such as TCP. Then, in the case of point to point transmission, one can argue that a selective acknowledgement procedure is always more efficient, transmission wise, because selective retransmission inserts exactly the level of redundancy that is actually needed.

But modern application often involve more than two parties. In their communication [1] to the 1995 Sigcomm conference Sally Floyd and her coauthors present their reliable multicast framework and explain why "repair request and retransmissions are always multicast to the whole group." A consequence of this architectural decision is that the overhead of retransmissions will grow with the size of the group. In section 3 we will detail this effect and show how it can be alleviated by packet level FEC.

The reduction in the packet level error rate may in fact be beneficial for point to point exchanges, when the transport protocol does not exhibit the near-perfect characteristics. This is the case for example of simple TCP implementations, which rely on time-outs to cope with certain forms of errors. We will discuss this aspect in section 4.

Even when perfect retransmission procedures are available, the correction of errors result in long transmission delays, specially when the application requires that data be re-ordered before processing. Section 5 explains how packet level FEC results in better delays and is thus suitable for delay sensitive applications.

Before exposing the advantages of packet level FEC, we will first examine how it can be implemented and used in practice. We will later re-examine the implementation problem after exposing the main advantages of the solution, showing how the level of redundancy can be adapted to the network configuration and status.

2 Packet level forward error correction

Bit level error correction operates on sequences of bits. It comes in two variations, block codes and convolutional codes. In block codes, a set of N input bits is completed by M redundancy bits. Out of the 2^{N+M} codes which can be encoded with $N+M$ bits, only 2^N are allowed. When the block is received, the redundancy is used to find out the allowed code which best matches the received pattern, thus removing errors if at least N bits out of $N+M$ were received correctly. In convolutional codes, each transmitted signal is a function of the input signal and of the history. Convolutional codes may be more robust than block codes, because they can use the redundancy which was spread over several consecutive blocks. They are also much more difficult to implement, at least in software.

In this paper, we will not consider the variants of packet level redundancy that can be built with convolutional codes. For practical reasons, we will only consider block codes where the transmission of N input packets is complemented by that of M redundancy packets. If at least N packets out of $N+M$ are received correctly, then all N input packets can be retrieved. If fewer than N packets out of M are received, we cannot gain advantage from the redundancy but we can at least retrieve the fraction of the initial N packets which made their way to the receiver.

Packet level correction is different from bit level correction, because it deals with straight packet losses, not with unpredictable bit values. It can thus be implemented very simply with a combination of *exclusive or* ($\oplus$) and *shift* ($\ll$) operations. If we note $X_{1..N}$ the input packets and $Y_{1..M}$ the redundancy packets, then we can obtain Y_j as a function of $X_{1..N}$:

$$Y_j = \oplus \left(X_i \ll (i^{j-1} - 1) \right)$$

The resources necessary for implementing this form of redundancy are memory and computing power. The sender uses M additional buffers to compute the redundancy, the receiver must dispose of $N + M$ buffers to use it in case of errors. The cost of computing each Y_j is approximately equivalent to one memory move per input packet. Senders and receivers will probably easily dispose of the required memory, but the computing cost has often been deemed excessive. A redundancy of form $N + 1$ may dobble the processing requirement of the best TCP implementations. We will thus only consider the redundancies of forms $N + 1$, $N + 2$ and $N + 3$.

For simplicity sake, we will assume that errors are independant. With the $N+1$, $N+2$ and $N+3$ redundancies, the perceived packet loss rate is a function of the initial loss rate ϵ and the number N. The resulting rates ϵ_1, ϵ_2 and ϵ_3 can be computed as:

$$\epsilon_1 = \epsilon \left(1 - (1 - \epsilon)^N \right)$$

$$\epsilon_2 = \epsilon \left(1 - (1 - \epsilon)^{N+1} - (N + 1)\epsilon(1 - \epsilon)^N \right)$$

$$\epsilon_3 = \epsilon \left(1 - (1 - \epsilon)^{N+2} - (N + 2)\epsilon(1 - \epsilon)^{N+1} - \frac{(N + 2)(N + 1)}{2}\epsilon^2(1 - \epsilon)^N) \right)$$

In each case, the amount of additional overhead is $M/(N+M)$. The question that we set up to solve is whether the gain of a reduced error rate is worth the pain of this additional overhead, as well as the cost of implementing the redundancy.

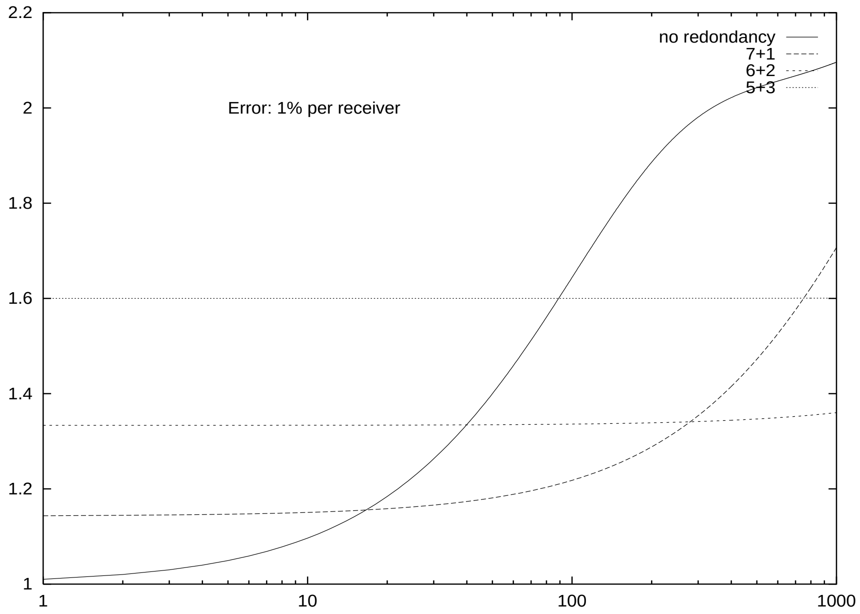

Figure 1: *Number of transmissions, as a function of the number of members in the group, the packet error rate being 1%*

3 FEC and large multicast groups

Transmission efficiency is often defined as the "goodput" ratio, i.e. the number packets that will be received and processed divided by the total number of packets that will be transmitted. According to this ratio, for point to point transmission, packet level redundancy is always less efficient than selective repeat procedures. In one case, we will pay an insurance and always send some redundancy in the hope that this redundancy may correct a transmission error. In the other case, we only send the redundancy after an error has occured, thus effectively minimizing the overhead. The situation changes however if we consider multicast transmission.

To demonstrate this effect, lets consider a very simple model. A source S transmit a stream of packets, for example a data file, towards a group of G receivers, using a multicasting service. We will assume that all transmission errors occur in the last leg of transmission, that the errors experienced by the receivers are independent and that each receiver has a probability ϵ of not receiving any given packet. This hypothesis is not entirely realistic because errors will also occur in the common paths between S and subsets of receivers and because receivers will experience different networking situations. We adopt it because it results in very simple computations, and also because the simplication does not render our results invalid. Multicast packets may be affected by errors at different stages of transmission, but it is very clear that the more receivers we add to the group, the largest the probability that some of them will loose any given packet.

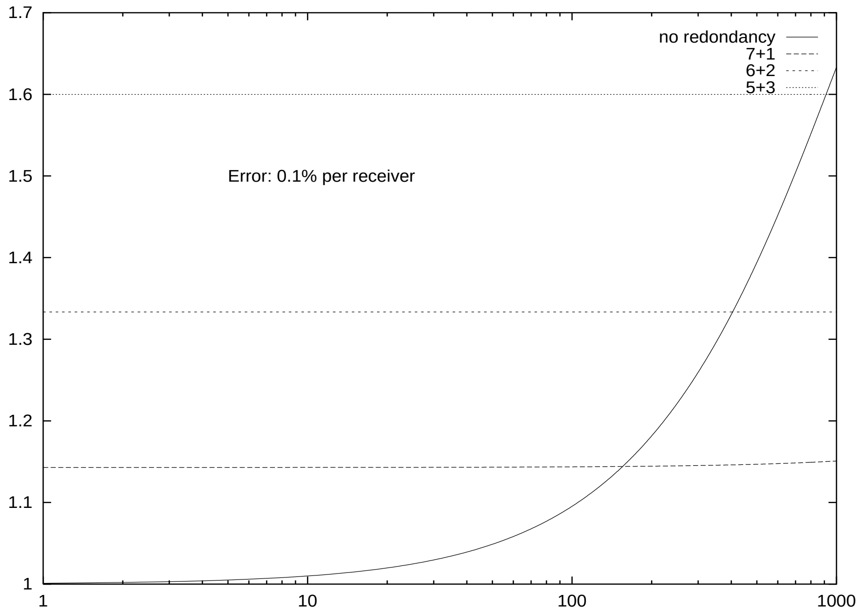

Figure 2: *Number of transmissions, as a function of the number of members in the group, the packet error rate being 0.1%*

The figure 1 shows the average number of time a packet has to be transmitted in order to be received by a group of size G, as a function of the size of the group. It takes into account both the initial transmission due to the redundancy scheme and the other retransmissions caused by the selective repeat procedures. As explained in [1] we assume that all retransmissions are sent to the whole group. In the figure 1 we assume that the packet loss rate is $\epsilon = 1\%$, a rate which is not uncommon in the internet today. The figure contains four curves, corresponding to the raw case, without redundancy, and to three variations of redundancy, namely 7+1, 6+2 and 5+3. These three levels are chosen arbitrarily, the rationale being that they only require the buffering of 8 packets, and are thus reasonably easy to implement.. When redundancy is used, ϵ is replaced by the lower value ϵ_1, ϵ_2 or ϵ_3, i.e.6.8 10^{-4}, 2 10^{-5} or 3.4 10^{-7} instead of 10^{-2}. For low recipient numbers, the gain of redundancy is not worth the increased overhead of even a $7+1$ scheme. But the number of transmissions increases with the number of recipients. As soon as this number reaches 18, the *no redundancy* curve passes above the $7+1$ limit, which will itself be passed by the $6+2$ curve when the number of recipient is larger than 300. The physical explanation is very simple. A packet will be retransmitted once if it lost by at least one receiver. The probability of this event is:

$$1-(1-\epsilon)^G$$

There may well be multiple retransmissions for a given packet, for example if a sufficient number of recipients missed the initial transmission: the common

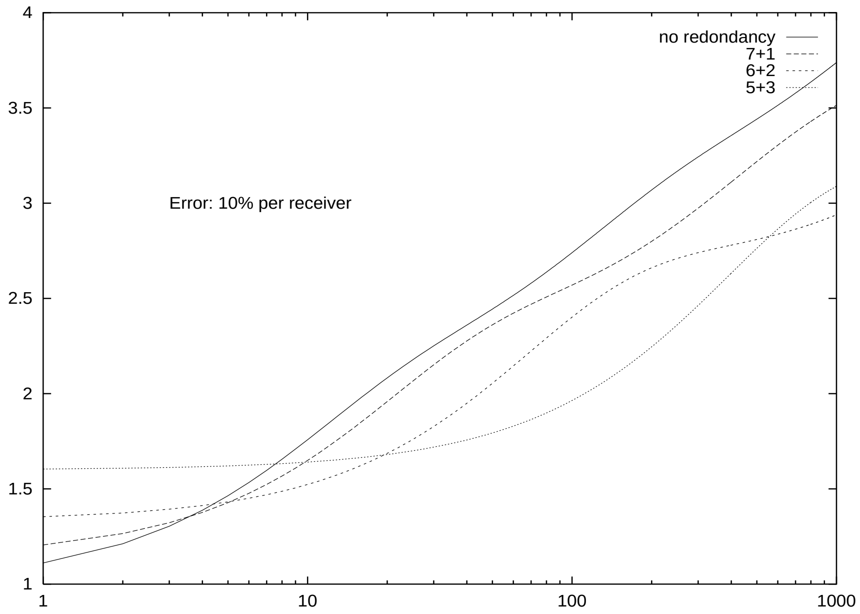

Figure 3: *Number of transmissions, as a function of the number of members in the group, the packet error rate being 10%*

assumption is that a single retransmission always suffice is too simplistic. The probability of this event increases with the size of the group. In fact, the probability that one given receiver requires exactly X transmissions of a given packet is:

$$(1-\epsilon)\times\epsilon^X$$

Alternatively, we can write that the probability of having fewer than X transmissions is:

$$1-\epsilon^X$$

In the case of multiple recipients, a packet will have to be retransmitted as long as it is not received correctly by all recipients. The probability of having fewer than X transmissions, for a group of size G, is:

$$(1-\epsilon^X)^G$$

We used this formula to compute the curves of figure 1, as well as those of figure 2, for a loss rate of 10^{-3}, and those of figure 3, for a loss rate of 10^{-1}. The comparison of these three figures shows that there is not one best level of redundancy. We have to balance the gain of reduced errors and the pain of increased overhead, and the trade-off is a function of the loss rate and the size of the group. This is further demonstrated by the curves of figure 4 which shows the efficiency of transmission as a function of the error rate, for a group of 100 receivers. We can distinguish four successive ranges, where each of the possible solutions chose be chosen. We may also observe the limits of packet level redundancy. If the packet

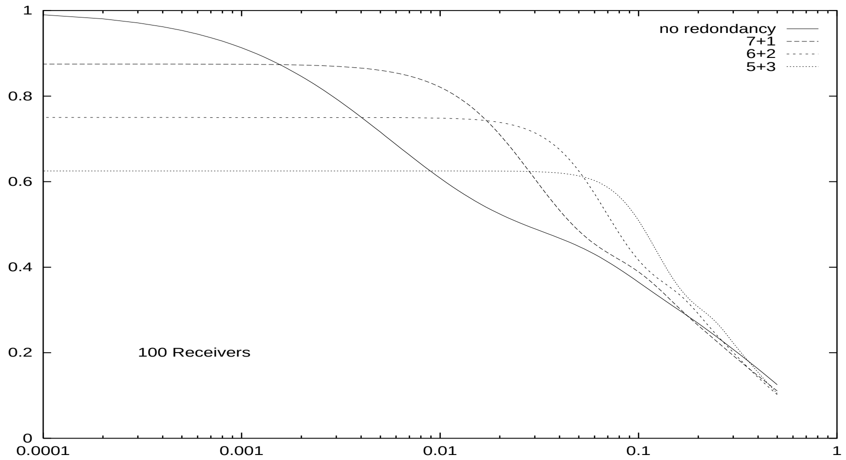

Figure 4: *Efficiency as a function of the error rate, for a group of 100 receivers*

loss rate becomes very large, say higher than 10%, even a redundancy of rate $N+3$ cannot result in satisfactory performances. This is also visible in figure 3.

Chosing the best level of redundancy depends however on the sender's objectives, as reducing the packet loss rate has two effects besides allowing a better goodput for file multicasting. It also reduces delay, and may be used to simplify the retransmission procedures.

4 FEC and simple transport protocols

In the previous section, we assumed that the transport protocol implemented fully selective retransmissions. In practice, implementations often fall short of this ideal. In classic implementations of TCP, once an error has been detected by a timer, a retransmission is triggered. The connection will remain inactive as long as this retransmission has not been acknowledged. During this waiting for retransmission, we could have transmitted a full window of W packets. This waiting time will occur after every error, thus at the end of every run of successful transmissions. As the length of such runs is $1/\epsilon$, we may define the efficiency of the procedure as the ratio of the length of the run over the number of packets that could have been transmitted in the absence of error, i.e., the length of the run plus the size of the transmission window:

$$\frac{1/\epsilon}{W+1/\epsilon} = \frac{1}{\epsilon W+1}$$

This formula does indeed not take into account the details of congestion control algorithms. It is in some sense an upper bound to the efficiency of classic TCP. We used it to compute the curves of figure 5, where we assumed a window size of 200 packets. These curves show clearly the usefulness of redundancy when the

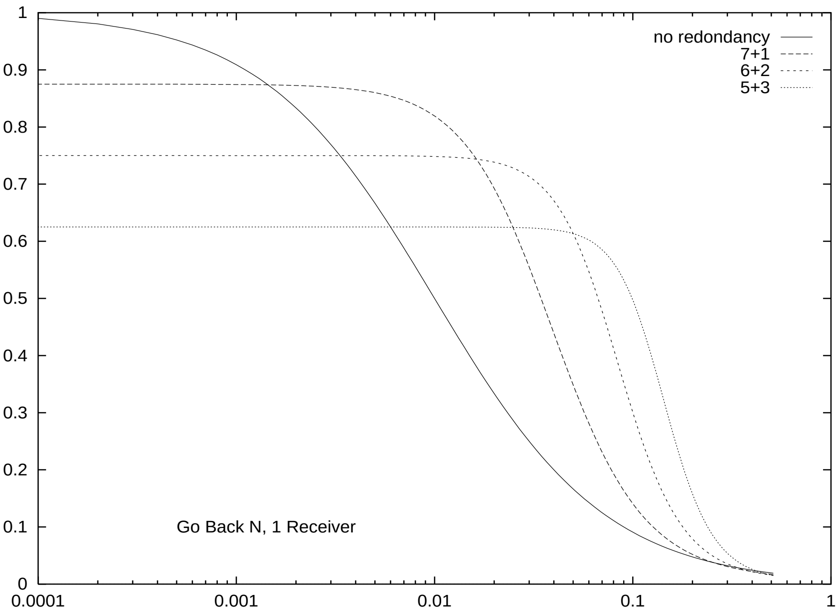

Figure 5: *Efficiency of classic TCP as a function of the error rate, for a window size of 200 packets*

error rate is larger that 10^{-3}. The threshold does indeed vary with the size of the window. An approximation of this threshold can be obtained by observing that $N + M$ redundancy is never justifiedif the goodput of straight transmission is better than the best that can be achieved with redundancy, that is if:

$$\frac{1}{\epsilon W + 1} < \frac{N}{N + M}$$

This inequality can be simplified to:

$$\epsilon W > \frac{M}{N}$$

By observing figure 5 we see that the converse is also true. $7 + 1$ redundancy is already almost justified when the efficiency of straight transmissions falls under 87.5%, $6 + 2$ redundancy is almost immediately justified when the efficiency of $7 + 1$ falls under 75%, $5 + 3$ redundancy is almost immediately justified when the efficiency of $6 + 2$ falls under 62.5%. We will use this observation in section 6.

Indeed, we know how to implement better procedures than classic TCP, using for example fast retransmission in case of duplicate acknowledgement. Matt Mathis, Jamshid Mahdavi, Sally Floyd and Allyn Romanow recently proposed a selective acknowledgement extension to TCP that would in theory results in an almost perfect utilization of the transmission resources [2]. However, their proposal establishes a balance between transmission efficiency and implementation requirements. As selective acknowledgement can only be advisory, Mathis et al have to rely on timers to solve errors which would not be cured by a first retransmission. This will occur when both the initial packet and the retransmission are lost, with a probability of ϵ^2. The waiting time of W packets will thus occur at the end of each run of successful transmissions or retransmissions. The length of this runs will be $1/\epsilon^2$, an upper bound of the efficiency will be:

$$\frac{1/\epsilon}{W + 1/\epsilon^2} = \frac{1}{\epsilon^2 W + 1}$$

We may thus foresee that packet level redundancy will be useful in the case of very large windows, even if we implement selective acknowledgements, when the product $W\epsilon^2$ is larger than M/N. In fact, we have also to take into account the need to allocate large resequencing buffers when waiting for selective retransmissions. This will be discussed in the next section.

5 FEC and resequencing delays

The delivery delay of a packet has three components, initial queuing before transmission, transmission and queuing at the recipient side. In order to simplify our demonstration, we will assume that the initial queuing delay is nil. We have seen above that the probability of having fewer than N transmissions for a group of size G is:

$$p(i < N, G) = (1 - \epsilon^N)^G$$

The retransmissions terminates once a copy of the packet has been correctly received. If we assume that the transport mechanism is very efficient, successive packets will be retransmitted independently of each other, hence may well arrive at different times. Many applications require that the packets be reordered before they can be successfully delivered. A packet which has been correctly received will thus be kept in a resequencing buffer, waiting for the successful retransmission of all preceeding packets.

In order to evaluate the resequencing delay, we will assume that the initial transmissions are regularly spaced. We will also assume that each retransmission takes the same delay as the transmission of a full window, "W". Figure 6 presents the variation of the delay as a function of the error rate for basic transmission, without redundancy, and also for three different forms of redundancy, $7 + 1$, $6 + 2$ and $5 + 3$. The unit of the resequencing delay, in this figure, is the round-trip delay itself. We have assume a transmission window of 200 packets, which would correspond to a T1 satellite network, or to a long distance T3 networks. Generally, the effect will be more intense with high values of W.

The figure is very expressive. The average resequencing delay soars when the error rate grows and so does the variance of this delay grows. The advantage of using FEC for delay sensitive applications is thus obvious. The maximum

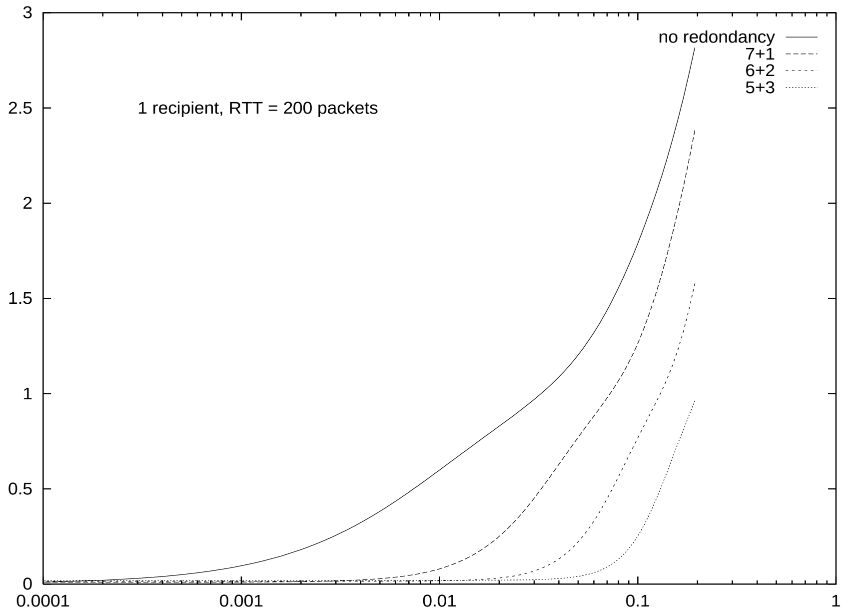

Figure 6: *Sequencing delay as a function of the error rate, for one receiver.*

redundancy rate of $5+3$ results in stable delays over a large range of packet loss rates. There are many applications that would be happy to pay this overhead, e.g. when user satisfaction matters more than network efficiency.

But this figure calls for another remark. When the resequencing delay increases, the memory requirements increase at the same time. TCP requires that the sender keep in memory a copy of all unacknowledged packets. The average number of acknowledged packets is equal to the sum of a transmission window, W, plus the average number of packets waiting to be resequenced. This second part is in fact proportional to the average retransmission delay. A consequence is that implementing selective retransmission on "long fat networks" is by no means free. Similar requirements will also occur at the recipient, which shall keep a copy of all packets waiting for resequencing. This may well explain why implementors have insisted on the "advisory" nature of selective acknowledgements: under certain conditions, the memory requirements of these systems may exceeds the number of buffers available.

A system that runs short of buffers cannot benefit fully from the theoretical efficiency of selective retransmissions. We are brought back to the imperfect efficiency that we studied in section 4. Using FEC in this conditions will stabilize the memory requirements, resulting in an improvement of the goodput.

6 Choosing the level of redundancy

All our graphs have shown that there is not one "best" level of redundancy. We already discussed this in section 3 and 4 but we limited our analysis to three simple cases, $7+1$, $6+2$ and $5+3$. Figure 7 shows a more systematic exploration of the problem. We assumed that the sender knew the number of recipients (100)

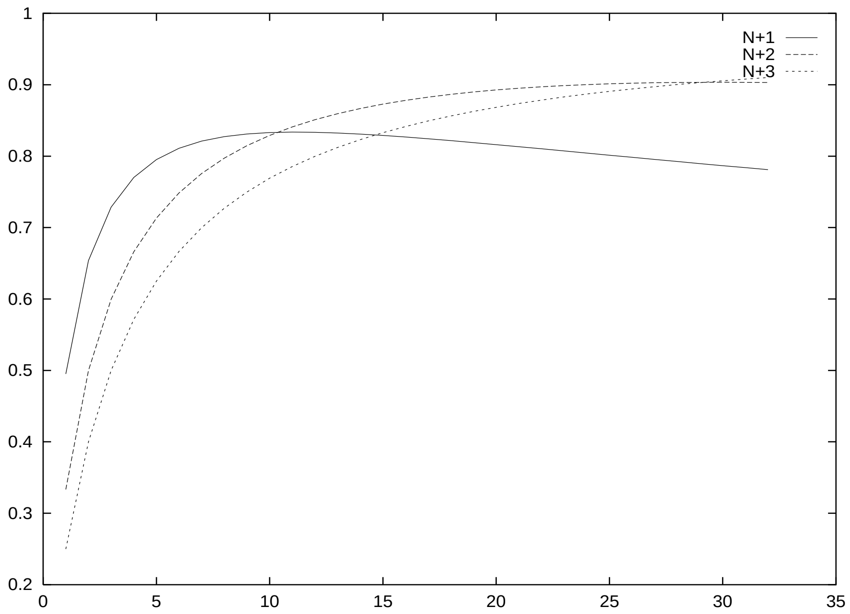

Figure 7: *Efficiency as the function of the size of the block, for 100 receivers and a packet loss rate of 1%*

and the loss rate (1%), and we plotted the efficiency of various level of redundancy, $N+1$, $N+2$ or $N+3$ as a function of the number N of "initial" packets in the block. The result is in fact quite obvious. If we can pick a block that is arbitrarily large, we should always use $N+3$ redundancy. If on the contrary there is a limit to size of the block, we may have to settle for less, or maybe for no redundancy at all. This trade-off will be a function of several factors such as:

- the sender's objective, efficiency or response time,
- the available memory, that may restrict the size $N+M$ of the redundancy block,
- the size of the multicast group,
- the observed loss rate,
- the delay$\times$banswidth product,

These parameters are likely to vary during the course of a transmission. There is thus a need to define an adaptive algorithm for chosing the redundancy level. The good news is that using FEC allow us to easily meter the basic error rate without necessarilly suffering from these errors. The semantic of acknowledgments and other "transmission reports" should be augmented to contain an "observed loss rate" parameter. This loss rate ϵ can be used to determine the necessary redundancy, for example with the help of precomputed tables.

Most packet losses in the current Internet are due to congestion. Algorithms such as slow-start have been devised to identify the throughput of the network's bottlenecks and to stabilize the transmission rate around this throughput [3]. The use of forward error correction should not result in an increase congestion, which suggests two implementation requirements:

- reduce the congestion window when redundancy is used,
- use the observe error rate to tune the congestion window.

A station which decides to use redundancy should not suddenly increase its transmission rate. The actual transmission window W'_c should be a fraction of the value W_c that was computed using the slow-start algorithm. Specifically, this fraction should be:

$$W'_c = \frac{N}{N+M} W_c$$

Because FEC will wipe out isolated losses, the station will not notice the first packet losses that are usually the warning signs of congestion. It may go on increasing its transmission rate until the congestion becomes massive. This undesirable behavior will be avoided if the receivers report the observed error rate and if the station reduces its congestion window whenever this rate increases.

7 Implementing FEC

This paper showed that packet level FEC can be implemented with benefits when transmitting reliably towards large groups, when using simple transport protocols and when requiring stable delays. Usual objections such as the concern for congestion may be overcomed by a careful tuning of the interaction between forward error correction and congestion control.

The purpose of our intellectual exercize was to demonstrate the interest of packet level FEC. We believe that we made this point. We have now to go back to the workbench and come out with a practical demonstration over the Internet. We have good hopes to complete this demonstration before August 1996.

References

[1] Floyd, Sally, Van Jacobson, Steven McCanne, Ching-Gung Liu, Lixia Zhang. *A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing.* Proceedings of ACM SIGCOMM'95, Computer Communication Review, Volume 25 Number 4, October 1995.

[2] Mathis, Matt, Jamshid Mahdavi, Sally Floyd, Allyn Romanow. *TCP selective acknowledgement option.* Work in progress.

[3] Jacobson, Van. *Congestion avoidance and control.* Proceedings of ACM SIGCOMM'88, Computer Communication Review, September 1988.