RMDP: an FEC-based Reliable Multicast protocol for wireless environments

Luigi Rizzo[a] Lorenzo Vicisano[b]
l.rizzo@iet.unipi.it *L.Vicisano@cs.ucl.ac.uk*
[a]Dip. di Ingegneria dell'Informazione, Univ. di Pisa, via Diotisalvi 2 – 56126 Pisa, Italy
[b]Dept. of Computer Science, University College London, Gower Street, London WC1E 6BT, U.K.

In this paper we present a Reliable Multicast data Distribution Protocols (RMDP), and discuss its performance. The protocol is based on the use of FEC techniques to drastically reduce the impact of independent losses for different receivers, which make ARQ-based protocols perform very poorly as the number of receivers grows. The protocol is well-suited to the use with mobile equipment because of its simplicity, robustness to losses, moderate demand for feedback, and scalability.

I. Introduction

Multicast/broadcast media have often been used to support computer communications, even on wired networks. The multicast nature of the medium, though, has not been fully exploited for a long time, because of the difficulty in extending this feature to wide area networks, and also because of the (relative) lack of applications in need of multicast support. The situation has changed in recent years, because of the increasing amount of information available in electronic form (e.g. through the Web), the widespread use of digital techniques in audio/video processing, and the advances in network technology. These technological developments have increased enormously the number of machines connected to wired or wireless network, thus able to access multimedia information. This in turn has created many new potential users of information services, making the use of multicast transport protocols an attractive approach for the distribution of popular data objects.

Several open problems still exist though, related to the scalability of the techniques used for multicast applications, especially those requiring reliable data delivery. In this paper we deal with one of these problems, namely the development of a reliable multicast protocol that is scalable to large groups of heterogeneous clients and suited to a wide range of loss rates. Such requirements are typical (although not exclusive) of a wireless environment, where users with fixed or mobile equipment access public information servers asking for the same data (e.g. newspapers, timetables) but possibly at slightly different times or with different loss patterns, due to mobility and communication errors.

The conventional approach to reliable multicast in computer networks relies on the retransmission on demand of lost packets. Scalability problems, that arise when the set of receivers becomes large, are generally approached by implementing a tree of nodes that share the burden of handling retransmissions [6, 9, 20], and by using ingenious strategies to avoid a phenomenon called "feedback implosion" [5, 6, 13].

The work presented in this paper significantly differs from existing implementations of reliable multicast, being based on the use of Forward Error Correction (FEC) techniques. Because of the encoding used to transmit data, successful completion of the data transfer in our protocol depends on the number of received packets, rather than on their identity. This feature enormously simplifies feedback handling, improves scalability, and increases tolerance to packet losses and client heterogeneity.

The use of redundant data encoding is common practice in telecommunication systems, where the encoding/decoding is generally performed through dedicated hardware. However, concerns on the computational overhead involved with the encoding and decoding process have often driven protocol designers away from this option, or suggested them to use hardware-based solutions [11].

Some papers have recently studied the use of FEC in computer communication [2, 8, 12], especially in link-level protocols [7]. But, excluding the simple XOR-based parity schemes [18] also used in RAID disks, few if any real implementations of FEC-based protocols exist, and recently proposed applications [17] or products [4] are heavily based on RMDP [16] and our congestion control algorithm [19].

In this paper we prove, through an actual implementation and its performance evaluation, that the data encoding/decoding can be implemented in software with the desired level of performance even on resource-limited equipment such as mobile computers and PDA's. Furthermore, we show that the use of FEC has significant benefits on the structure of the protocol. It not only provides additional protection from packet losses, but it also greatly enhances the scalability of the protocol and reduces the amount of feedback from receivers. This in turn gives further benefits in efficiency of use of the communication channel, and reduction of transfer times and power consumption on the receiver.

The paper is structured as follows. We start by describing the problem of reliable multicast for large and heterogeneous groups of receivers. We continue by giving some detail on the principle of operation of erasure codes, that are used in our protocol to achieve reliability. Section III describes our protocol, whereas the evaluation of its performance is presented in Section IV. We conclude by discussing some details related to our implementation of RMDP.

II. Reliable multicast

Reliable multicast (RM) protocols are in charge of distributing the same data object (generally split into a number of data packets) to a set of receivers, with some kind of guarantee on the delivery process. Depending on the application, the protocol might be required to deliver packets in a certain order, or to handle multiple objects from different sources with specific ordering constraint. In this paper we present a protocol that guarantees the *eventual* delivery, to all successful receivers, of the same object, with no special ordering or timing constraints in the delivery of individual packets. Such a guarantee is suitable for a large class of applications where receivers are independent, so they do not need mutual synchronization.

A. ARQ

RM protocols must face two distinct problems: provide reliable data delivery, and accomodate more than one receiver. Reliable data delivery is a typical problem also in unicast communication, and it is generally solved by having the receiver explicitly (by means of negative acknowledgements) or implicitly (using positive acknowledgements and timeouts) request the retransmission of missing packets. The retransmitted packets are often called *repair* packets. This technique, called ARQ (Automatic Retransmission reQuest), is extremely inexpensive from the computational point of view, and is largely used in communication protocols on wired networks.

Unfortunately, ARQ scales very badly to large sets of receivers. For a packet loss rate (PLR) of p, and a set of r receivers experiencing independent losses, the probability that every single data packet needs to be retransmitted is $P(r,p) = 1 - (1-p)^r$, and this value quickly approaches unity as r gets large. This also translates in a high average number of transmissions per packet, as will be shown in Figure 4. The situation does not improve if losses at different receivers are partly correlated: when the group becomes large, so does the number of subsets of receivers with uncorrelated losses, and the same phenomena appear, only at a different scale.

Scalability problems also exist in handling feedback from the receivers: in fact, the sender must handle distinct error reports from every receiver, resulting in an average of rp reports per transmitted packet if plain ACK/NACK is used. Such a traffic can easily saturate the feedback link or the sender itself. A technique exists, known as NACK suppression [6, 13] that combines negative acknowledgements and multicast feedback channel, and allows to contain the amount of feedback. Receivers in the need of a retransmission schedule a multicast NACK transmission at some random time in the future, and listen to the feedback channel for NACK requests coming from other receivers. The locally scheduled NACK request will be delayed if another NACK is received, and canceled if a repair packet arrives.

Other solutions, used in combination with ARQ to contain the amount of feedback, consist in using a 'local recovery' approach, where retransmissions are handled by other receivers close to the ones missing data. Apart from the complexity deriving from the establishment of hierarchies of receivers, these solutions rely on receivers to buffer incoming data (even beyond the needs of applications) and transmit repair packets. This approach is clearly not applicable in a wireless/mobile environment where communication is highly asymmetric and receivers can move frequently.

In a wireless environment, ARQ has a second major disadvantage, consisting in the requirement of a bidirectional communication link. On most wired networks the feedback channel comes for free, but on wireless networks the transmission of feedback from the receiver can be expensive, either in terms of power consumption, or in usage fees for the use of a communication infrastructure.

B. FEC

A different approach to improve reliability is often used in telecommunication systems, and is based on the use of pure FEC (Forward Error Correction) or hybrid FEC+ARQ techniques [3]. Pure FEC relies on the transmission of redundant data (generated by a suitable encoder) that allows the receiver to reconstruct the original message even in presence of some communication errors. Hybrid FEC+ARQ is typically implemented by reducing the initial amount of redundancy, and sending repair packets only on demand as in ARQ-based protocols.

As an example of FEC, one can feed k source data packets to an *encoder*, which produces $n > k$ encoded packets in such a way that any subset of k encoded packets allows correct reconstruction of the source data at the receiver. By choosing suitable values of n and k, the residual error rate after decoding can be made arbitrarily small. This feature can make a feedback channel unnecessary, and is what makes FEC attractive in telecommunication systems, even if intended for unicast communication.

Implementing FEC is computationally expensive, since the entire data stream must be processed to produce the encoded packets, each one conveying information on a number (possibly as large as k) of source data packets. This is not a concern in telecommunication systems, where the encoder/decoder is usually implemented as a dedicated piece of hardware and is usually much cheaper than having a feedback channel. But in computer communications, the feedback channel is often available and implementing FEC means a noticeable overhead for the main processor.

In multicast protocols, however, the use of FEC techniques has completely different motivations. The encoding is mainly used to remove the effect of independent losses at different receivers: thanks to the encoding, as long as a receiver collects a sufficient number of different packets, reconstruction of the original data is possible independently of the identity of the received packets. This makes protocols scale much better irrespective of the actual loss pattern at each receiver [8, 12]. As an additional bonus, the dramatic reduction in the residual loss rate (after decoding) largely reduces the need to send feedback to the sender, thus minimizing the use of the uplink channel, and simplifying feedback handling.

C. Erasure codes

Communication systems are typically affected by *errors*, i.e. incorrect data in unknown locations. In computer communications, such errors are generally handled at a low level in the protocol stack, and either they are corrected, or cause the removal of the whole corrupt packet. Thus, upper protocol layers will mainly have to deal with *erasures*, i.e. missing packets in *known* locations.

The computations necessary for erasure recovery are slightly simpler than those for full error recovery. An (n,k) *block erasure code* (or, the corresponding encoder) takes k source packets and produces n encoded packets in such a way that any subset of k encoded packets (and their identity) allows the reconstruction of the source packets in the decoder (Figure 1). A code is called *systematic* when the encoded packets include the source packets in clear. Systematic codes

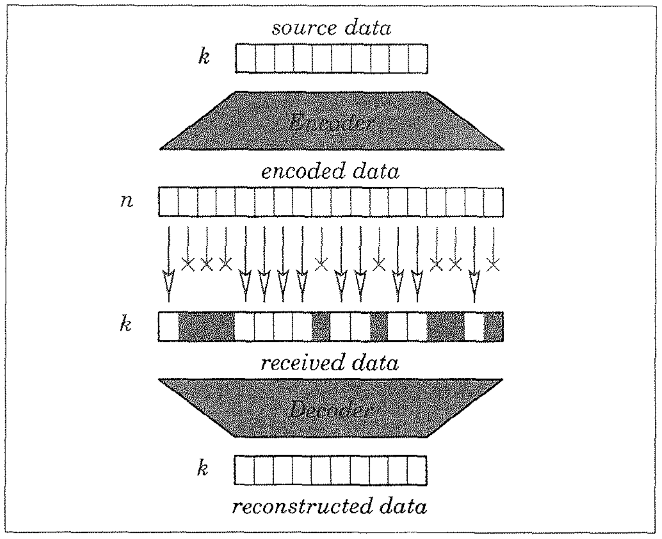

Figure 1: A graphical representation of the encoding and decoding process

System	CPU/Speed	c_d (MBytes/s)
Sun Ultra1	UltraSPARC 167 MHz	21.289
DEC	Alpha 255 MHz	12.577
HP715/100	PA7000 100 MHz	10.308
PC/FreeBSD	Pentium 133 MHz	9.573
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 1: Speed of the encoding/decoding process on various architectures

are much cheaper to decode when only a few erasures are expected; besides, they might allow partial reconstruction of data even when fewer than k packets are available.

In some cases, erasure codes are trivial to build. As an example, an $(n, 1)$ code is simply implemented by retransmitting n times the only source packet, and a $(k+1, k)$ code can be implemented by XOR'ing the k source packets [18]. For generic n and k, erasure codes can be constructed and studied basing on the properties of linear algebra over finite fields. A complete discussion of the subject can be found in [3], while [11] and especially [15] give a detailed description of erasure codes more closely related to their implementation.

The code we use, called Vandermonde code [15], interprets k source data as the coefficients of a polynomial P of degree $k-1$. As the polynomial is fully characterized by its values in k different points, we can produce the desired amount of redundancy by evaluating P at n different points. Reconstruction of the original data (the coefficients of P) is possible as soon as k of these values are available. In practice, the encoding process requires multiplying the original data by an $n \times k$ encoding matrix G, which happens to be a Vandermonde matrix. The decoding process requires the inversion of a $k \times k$ submatrix G' taken from G, and the multiplication of the received data by G'^{-1}. By simple algebraic manipulation, G can be transformed to make its top k rows constitute the identity matrix, thus making the code a systematic code.

The actual implementation of an erasure code is complicated by the fact that, using ordinary arithmetic, the values of the polynomial might require more bits to be represented than its coefficient. This expansion is prevented by doing computations in a finite field, or *Galois Field* [3]. Despite the lack of native support for finite field computations on ordinary processors, basic operations can be often implemented efficiently using XOR and table lookups.

As a consequence, the computations necessary to implement an erasure code involve just the solution of linear systems. The (amortized) decoding speed can be approximated as c_d/l, where c_d depends on the speed of the system, and l is the number of non-source packets used in the process (l is upper bounded by k, but the use of a systematic code can make l quite small). The speed of the encoding process is roughly c_e/k, where c_e is slightly larger (10-20% in most cases) than c_d.

Table 1 presents the value of c_d for a wide range of different systems, ranging from fast workstation to some very old and slow system which we believe to be comparable (if not slower) to modern PDA's. These values have been computed using our portable C implementation of the encoder/decoder [15] with no special optimizations. The algorithm uses mostly table lookups, so FPU's are not helpful, and performance on fast machines is limited by the memory bandwidth (for optimized implementations, the constants c_d and c_e could approach 1/5 of the memory bandwidth). Since both the encoder and the decoder essentially perform repeated matrix multiplications, they are very cache friendly. Furthermore, the memory footprint of the encoder is very small – the tables typically take about 64KB, which can be easily reduced to about 1 KB on very small memory systems.

Table 1 is not meant to present speed records for software encoding/decoding, but rather to give an idea of the range of values achievable for a large class of systems. Even by scaling the figures to account for reduced CPU usage or large values of k, these results show that software-based encoding/decoding is already feasible on nowadays systems. The speed of operation is well balanced with the typical communication speeds for each class of system. Processing power for doing the encoding/decoding is likely to be available on mobile equipment: developments of the cellular phone industry have in fact produced DSP's operating at 20..100 MIPS and costing less than US$10. There processors can run the encoding/decoding algorithms at high speed, and their inclusion on mobile equipment (if not already present, e.g. for performing audio processing functions) will have a very low impact on the overall costs. The obvious advantage of using software rather than hardware based solutions is that we can ride the technology curve and take advantage of the performance improvements of general purpose CPUs.

III. The RMDP protocol

RMDP is a hybrid FEC+ARQ protocol for reliable multicast data distribution to (potentially large) sets of receivers. In RMDP, the FEC encoding is used to improve the behaviour of the protocol in presence of large groups of receivers, and to reduce the amount of feedback from receivers. ARQ is used to deal with those cases where the default amount of redundancy does not suffice to complete reception.

We assume that the data object to be transmitted is a *file*, identified by a unique name, say its Uniform Resource Locator (URL). The file has a finite size, and is split into packets of s bytes each. RMDP uses an (n, k) encoder with $n \gg k$ to generate packets to be transmitted[1], and assumes the existence of a multicast network which provides unreliable – but efficient – delivery of data packets. This network can be as simple as an Ethernet segment, or a wireless cell, or as complex as the MBone (the multicast infrastructure on the Internet). Since the network might be able to carry multiple communications, we will convoy the packets of our interest onto a specific logical channel. In the case of the Internet, this channel will be identified by a multicast address/port pair; for other carriers, e.g. data over the Vertical Blanking Interval in TV broadcast channels [14], the channel might be a specific Teletext "page".

RMDP is not restricted to the use on wireless networks. As a matter of fact, our prototype implementation of RMDP [16] was initially targeted to reliable file distribution on the MBone. Depending on the actual choice of some of the operating parameters, the protocol can be tuned to provide effective operation on different types of networks. This paper will focus on operation on wireless networks where downlink communication is cheap, whereas uplink communication is expensive and must be minimized.

A. Protocol description

In the description of the protocol we uses the following parameters:

f	unique identifier of the file being transmitted.
l	total file size, in packets. For simplicity assume $l \le k$, this restriction will be removed in the next section.
D	default expansion factor, i.e. amount of redundant data sent unconditionally. $D = 1$ means that repair packets are only sent on demand, as in a pure ARQ protocol. Increasing D reduces the need of feedback from the receivers.
τ	inter-packet transmission time. Transmission rate in RMDP is fixed[2] and set by the sender, and equal to s/τ bytes/s.
c_s	count of packets left to send at the sender. The sender set c_s when requests arrive, decrements the count at each transmission, and stops transmitting when $c_s = 0$.
c_r	count of packets received. Reception is complete (and the decoding procedure can be run) when $c_r = l$.
p_i	i-th packet produced by the encoder

[1]The condition $n \gg k$ does not mean that RMDP sends data with unconditionally high redundancy, but is only used to reduce the chance that the same packet is received twice.

[2]In principle, RMDP could operate at variable speed adapting to the requests from the receivers, and a previous prototype of the protocol [16] did support this feature.

In RMDP there are only two message types:

- **R**$[f, c_r]$ is sent by a receiver to start a new transfer ($c_r = 0$), or extend an ongoing transfer ($c_r > 0$). Note that the identity of the receiver is not known to the sender.
- **S**$[f, l, c_s, i, p_i]$ is multicast by the sender and contains one data packet plus some information on the sender's state[3].

For simplicity we omit the initial exchange of the protocol where the URL is translated in the values (channel, unique id, etc.) necessary to the receiver to collect data packets. This exchange is not strictly part of the protocol since the mapping could be acquired by external means. Otherwise, the protocol can be extended to transmit, in response to a new request, the required information.

SENDER

```
• initialization:
    $c_s \leftarrow 0; i \leftarrow 0$

• receive R$[f, c_r]$ :
    $c_s \leftarrow \max(c_s, D \cdot (l - c_r))$;
    if ($c_s > 0$ and no transmissions pending) then schedule a
    transmission immediately fi.

• transmission time:
    send S$[f, l, c_s, i, p_i]$;
    $c_s \leftarrow c_s - 1; i \leftarrow (i + 1) \bmod n$;
    if ($c_s > 0$) then schedule a transmission after $\tau$ sec. fi.
```

RECEIVER

```
• initialization:
    $c_r \leftarrow 0$;
    schedule a request immediately;

• request time:
    send R$[f, c_r]$;
    schedule a request in interval $T_R$ ;

• receive S$[f, l, c_s, i, p_i]$:
    cancel any pending request ;
    if ($p_i$ is not a duplicate) then store $P_i$; $c_r \leftarrow c_r + 1$ fi ;
    if ($c_r = l$) then decode and exit fi ;
    schedule a request in interval $T_C$.
```

Figure 2 shows the evolution of c_s at the sender as multiple receivers join the group. The transmission of the initial request sets the count to Dl every time a new receiver arrives, thus extending the transmission and providing additional loss recovery for already active receivers.

To make feedback suppression work properly, requests for missing packets are not scheduled at fixed times (which would

[3]The overhead for sending f, l, c_s, i is minimal. Additionally, some of these parameters are constant for the whole transfer and can be easily compressed.

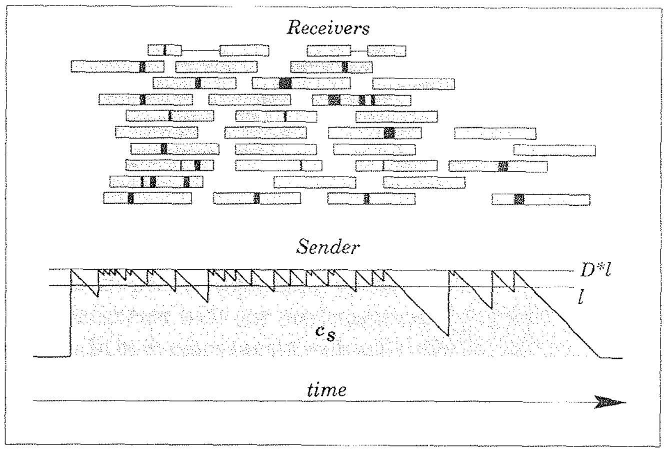

Figure 2: Evolution of c_s at the sender while receivers join and leave. The rectangles represent different receivers joining a session, with black areas indicating lost packets at each receiver.

cause feedback storms), but at random times within appropriate intervals T_R and T_C. The details on how to determine these intervals are discussed in Section IV-C.

We remark that in RMDP the decision on whether reception is complete can be taken with very simple computations and without performing any decoding. The receive phase only consists in collecting packets and storing them, and need not be slowed down by the CPU-intensive decoding.

B. Adapting the protocol to large files

With our encoder, the cost for encoding (and decoding) each packet grows linearly with k. This suggests the use of small values (say 32..64) for this parameter. Coupled with packet sizes of about 1 KB, it follows that our assumption of $l \leq k$ does not hold for medium-large sized files. Two approaches are possible to deal with this problem.

The first one relies on the use of better codes, such as the Tornado codes presented in [10]. These codes have better asymptotic performance than our Vandermonde code, so they can be used even for large values of k, as shown in [4]. Tornado codes are non deterministic, meaning that the number of packets necessary for a successful decoding varies, and the decision to stop receiving can only be taken by performing a partial decoding of the data stream. This could be problematic on resource-limited systems. Another major obstacle to the use of large values of k is the need to keep in main memory $O(k)$ packets for encoding/decoding purposes. Small devices may easily hit memory constraints even with fast codes.

The second approach, used in RMDP, consists in splitting the whole file in B slices of at most k packets each, applying the FEC encoding to each slice separately, and interleaving transmissions by taking one packet from each slice (see Figure 3). The source of inefficiency in this approach lies in the fact that some receivers can collect more than k packets per slice. While not as effective as using a large k, the interleaving gives good robustness to bursty losses, and permits the transmission of large files with reasonable encoding costs and a good channel utilization.

The protocol does not change significantly in presence of

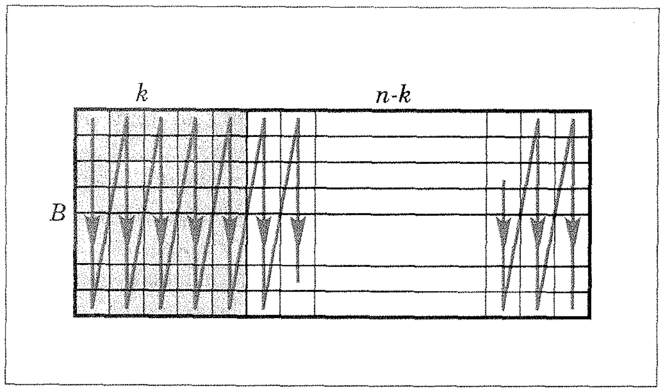

Figure 3: Transmission order of packets when $l > k$.

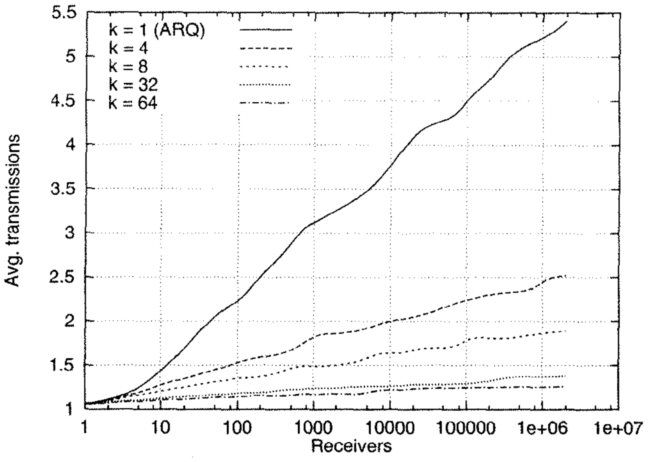

Figure 4: Average number of transmission per packet, for pure ARQ (k=1) and hybrid FEC+ARQ protocols for different slice size. In this simulation we have assumed independent, uniformly distributed losses with 5% loss probability.

multiple slices. In particular, the sender only needs to keep an additional index to the current slice. On the receiver, the decision on how many packets are necessary to complete transmission must be taken by considering the slice with the largest number of misses (the additional state required at the receiver is only one counter per slice, and the test is straightforward).

The data organization also does not pose significant problems, since the receiver simply stores received packets and passes them to the decoder, one slice at a time, when each slice is complete. Thus, it is not necessary that the receiver keeps the whole file in memory at once: optimal performance can be achieved by just bringing in main memory one slice at a time. Inability to store the whole file in memory could cause some performance degradation in the sender, since subsequent transmission operate on different slices. This problem can be simply overcome by generating several packets per slice at once (but still preserving the interleaving transmission order) so that the I/O costs can be amortized.

IV. Performance evaluation

Throughput and goodput are commonly used metrics to measure the goodness of a protocol. The throughput measures the speed of the data transfer, whereas the goodput is a measure of efficiency, being computed as the ratio between useful packets and received packets. ARQ-based protocols (especially in the unicast case) aim at maximising the throughput while achieving a goodput as close as possible to unity (these are somewhat contrasting requirements since keeping the goodput high means avoid the transmission of duplicates, and thus use longer timeouts). In RMDP, throughput is constrained by the transmission rate at the sender and by the loss rate, and both are factors which the protocol does not (or cannot) control. So the most interesting metric in our case becomes the goodput.

An additional concern, in multicast protocols, is the scalability in presence of large sets of receivers. Performance indexes for scalability are the average number of transmissions per packet, and the effectiveness of the feedback suppression mechanism.

It is convenient to start the analysis of RMDP by looking at the effects of the use of encoded packets for repairs. This will show that it is possible to study other aspects of RMDP's behaviour almost independently of the number of receivers, which simplifies the analysis of the protocol.

A. Scalability gain from the use of FEC

For small transfers ($l \leq k$, i.e. the file fits into a single slice) the goodput of the protocol is unitary for *all* receivers, since all received packets will be useful to any receiver (the condition $n \gg k$ guarantees that no duplicate packets are received for all but the most extreme loss patterns). This nice property matches the performance that ARQ-based protocol can only achieve in unicast communication. Goodput degradations for large files are discussed in the next section.

Regarding the scalability of the protocol with large groups of receivers, RMDP resembles generic FEC+ARQ protocols that have been studied by other researchers [8, 12]. Figure 4 shows simulation results for the average number of transmissions per packet, $\eta(r, k)$, for a generic FEC+ARQ protocol[4] with all receivers starting simultaneously. The slope of the curve depends on loss rate and distribution, group size r, and the slice size k. The curves in Figure 4 have been computed assuming $D = 1$, i.e. repairs are only sent on demand. The effect of $D > 1$ is very simple to account for, since $\eta(r, k, D) = \max(D, \eta(r, k))$.

The graph can be read as a measure of the scalability of the protocol for different size of the FEC block (or slice, in RMDP). The case $k = 1$ effectively corresponds to a pure ARQ protocol. The graph clearly shows that even small values of k have a dramatic effect on the scalability of the protocol, to the point that we can effectively neglect the group size in our evaluations.

B. Goodput for large files

In RMDP, large files are split in multiple slices of fixed size. This would not change the goodput of the protocol if precise feedback (i.e. the exact count of remaining/received packets) was used in **S[]** and **R[]** packets. However, we chose not to use this solution because it would increase the size of packets, and reduce the chance for merging/suppressing feedback. The use

[4]The ondulations in the curves are not an artifact of the simulation, see [12] for a more thorough discussion of these curves.

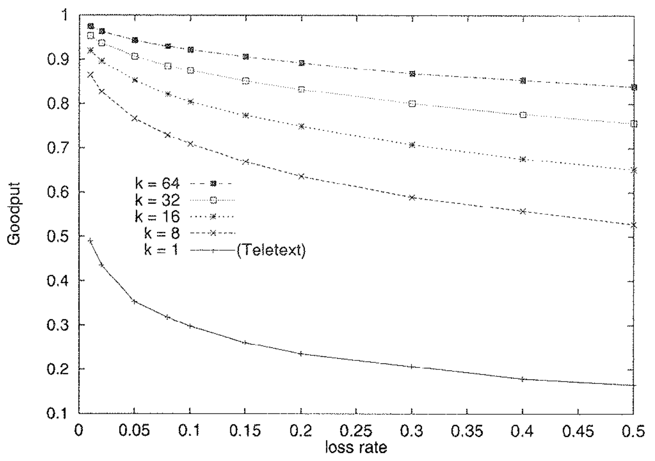

Figure 5: The goodput of the protocol for random losses with different loss rates and values of k. The file size is 2048 packets.

of imprecise feedback (relying on a single packet count) reduces the goodput, because a single missing packet may cause the transmission of a whole "column" of B packets, one per slice. The loss of efficiency is tolerable, as we show in the following.

At low loss rate, the loss of goodput can be quantified as $1/(k+1)$ (intuitively, the reason is that a single loss will cause an additional transmission of B packets, or $1/k$-th of the file size). For increasing loss rate, the goodput can be computed analytically in some simple cases, e.g. with uniformly distributed random losses. In this case, the following equations give the probability $P(x)$ that a file made of $l = Bk$ packets can be reconstructed with the transmission of x packets (obviously $x \geq l$):

$$P(x) = P_0(\lfloor x/B \rfloor)^B$$

where

$$P_0(x) = \sum_{i=k}^{x} p(i) \text{ and } p(x) = \left(\begin{array}{c} k \\ x \end{array} \right) (1-p)^k p^{x-k}$$

Analytical computations of the goodput are only partially useful, since the actual distribution of losses strongly influences the results, and depends on a wide range of factors which is hard to model. Random uncorrelated losses are in fact the worst loss pattern for our encoding. Burst losses are dealt with much more efficiently because they are spread over a number of different blocks. As an extreme case, the goodput with long bursts of missing packets can be as high as $k/(k+1)$. Also, for a given file size, larger values of k give better performance.

Figure 5, computed using the $P(x)$ defined above, shows the goodput for random uncorrelated losses, for a loss rate p and values of k in the range 1..64. The file size is 2048 packets. As shown by the graph, with the exception of very small values for k, the goodput remains within an acceptable range even in presence of substantial losses.

The bottom curve ($k = 1$) is not of practical interest for RMDP, but it is shown to illustrate how bad performance can be without precise feedback, and with recovery implemented by retransmitting all packets cyclically. It is worthwhile mentioning that such a poorly performant protocol is in widespread use indeed: Teletext data is in fact transmitted in this way by most TV broadcasters in Europe. Teletext information [1] is made of a number of *pages* which are similar to our files. In many cases one page fits in one "packet" (1000 bytes), but there are some noticeable exception where a single logical unit (e.g. stock quotes) is split into a large number (e.g. 30 or more) of packets which are transmitted cyclically.

We have run simulations with mixed loss patterns, modeled by a Markov process alternating bursts and independent losses, and experiments with our implementation of RMDP over the MBone. In both cases (again with a file size of 2048 packets of 1 KB each), the goodput values were very close those shown in Figure 5. This is a result of the moderate length of bursts, both in simulations and in real tests.

The loss of goodput discussed in this section should be considered in the right perspective. For first, it is not an intrinsic limitation of RMDP, but just a side effect of the use of a rather simple coder (but with the advantages of its very small memory footprint). Secondly, from the sender's point of view, the efficiency is generally much higher: in fact, a packet is useless only if it is useless for all receivers, and the probability of this event decays exponentially with the number of receivers experiencing uncorrelated losses. In practice, the only source of inefficiency at the transmitter are packets sent after the last receiver has completed reception.

C. Feedback suppression

The last aspect of scalability for multicast protocols consist in the avoidance of "feedback storms". The general approach consists in spreading the generation of feedback from receivers over a suitable interval, and making sure that early responses effectively cancel the generation of later ones.

In RMDP we use a similar approach, with additional advantages coming from the use of imprecise feedback and from merging **R**[] messages even if they have a different c_r count. Receivers will send a request for more packets at the expiration of a timeout, computed by picking a value at random in a suitable interval every time a packet is received or a request is sent. The rules used to compute timeouts are the following:

- when a packet arrives and $c_s + c_r < l$, the reception cannot be completed unless some receiver sends a request for more packets to increase c_s. The timeout is chosen within an interval T_C covering the last part of the scheduled transmission (and making sure that the interval has some minimum width to achieve an effective spread of receivers).

- when a packet arrives and $c_s + c_r \geq l$, no requests are necessary, unless some packet gets lost. Thus, the interval T_C is moved after the end of the scheduled transmission, just as a safety measure in case all subsequent packets get lost.

- when a timeout expires, a request is sent and a new timeout is scheduled within an interval T_R chosen using a truncated exponential backoff policy, much like the one used for scheduling Ethernet or TCP retransmissions. This is really necessary only when a single receiver remains active.

Such a mechanism, evaluated by simulation, performs extremely well. This was expected, since the number of requests is contained by a self-stabilising behaviour of the protocol itself:

> when the number of receivers is very large, the frequency of new clients joining the session is proportionally high, and thus the chance that the transmission terminates is low.

In the basic protocol description, each receiver issues a request when it joins the group. That might rise the concern of an implosion of requests when the group size becomes huge and receivers start at approximately the same time. From the point of view of the protocol this phenomenon is irrelevant: the worse that can happen is that some of these requests get lost, but since all requests are equivalent, the protocol will still be able to proceed at full speed. Of course, we want to protect the network from such phenomena. When there is the risk of such a problem, a simple and effective solution consists in canceling the initial request and instead scheduling a timeout in the near future. Any transmission before the timeout expires will cancel the timeout and provide sufficient information to the receiver to start operation.

We conclude the analysis on scalability with some considerations on what can be the expected sizes for the set of simultaneous receivers. The effectiveness of using multicast in place of unicast and adding the FEC overhead depends in fact on the number of overlapping receivers. Receiver aggregation is a statistical parameter influenced by the type of application and by the actual object being transmitted. If the number of simultaneous receivers is too small, the server itself can force the aggregation delaying the start of the transmission waiting for more receivers to join the group.

Group sizes depend heavily on the type of information being transmitted. For local-area applications, the number of simultaneous receivers is generally low, for two reasons: the small number of potentially interested users, and the high transfer rate which makes the transfer time short. Even for wide-area applications, it is unlikely that group sizes will be exceedingly large, basing on the fact that these files are available for much longer than the time necessary for the download, thus effectively spreading the set of receivers over the whole interval. In the most extreme situations reported in the literature [17], such as the release of a new software distributed over the network, or the electronic distribution of newspapers, it will be rare to have more than a few thousands overlapped receivers.

There is only one case in which the number of different receivers may become very large, and such a case is the live broadcast of popular events. There, the number of potential simultaneous receivers can grow very large but because of a number of cooperating reasons including the popularity of the event and the limited temporal interval during which data can be received. This is however a case where reliability is rarely required, so we do not foresee the application of a reliable multicast protocol to groups of millions of receivers.

D. RMDP on unidirectional channels

Albeit RMDP makes very little use of the the uplink channel, in some cases it might be desirable to completely avoid its use. Obviously, no feedback means that we cannot guarantee that

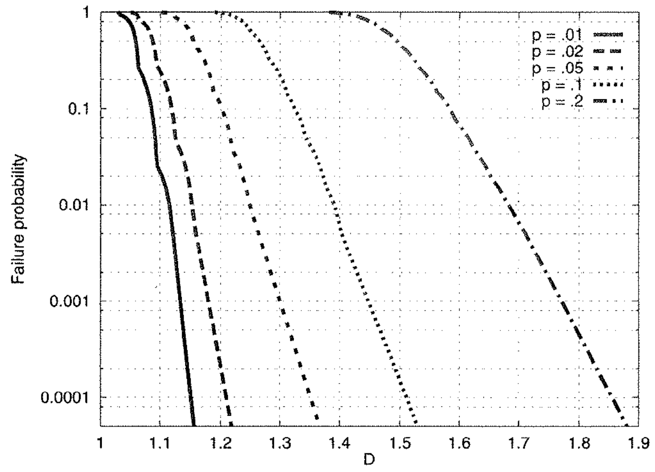

Figure 6: The probability of failure in RMDP for different values of the parameter D.

all receivers will be able to complete reception – we can only try to tune the protocol's parameters to minimize the number of "unsatisfied customers". The latter, in turn, should be prepared to sustain occasional failures in exchange for the savings deriving from not having to send feedback (and thus, not having to implement the uplink channel).

To achieve this goal, the sender needs to transmit more packets than those strictly necessary to complete reception in absence of losses. The amount of redundancy should be chosen to make the probability that a receiver has to send a continuation request so small to be negligible for practical purposes. As usual, this probability is affected by the actual loss rate, the number of receivers (because new requests will cause a file to be transmitted for a longer time) and the value of the parameter D of the protocol. The worst case corresponds to the presence of a single receiver, when the transmission of a file involves the generation of exactly $D \cdot l$ packets after the initial request. Figure 6 shows the failure probability for a file size of 2048 packets and $k = 32$, for different loss probabilities and values of D. From the graph we see that, even in presence of substantial losses, values of D between 1.5 and 2 will make the failure probability (hence the need to send a continuation request, or abort if no uplink channel exists) extremely low.

The graph can be read in two ways. If we consider the case of a single receiver (or multiple receivers starting at the same time), we can think of D as the amount of redundancy that must be used unconditionally by the sender. As an example, this situation might occur when a sender decides to transmit a file (e.g. the latest news, or stock quotes, etc.) at a certain time even in absence of explicit requests. In this case receivers will be synchronized, so the sender can set the amount of redundancy (i.e. how long to continue transmission) depending on the expected number of receivers and the acceptable number of failures (i.e. receivers that will be unable to complete reception in absence of a continuation request).

Another way to read the graph is to consider the presence of multiple receivers (see Fig. 2) starting at different times. This could be the case of users that access a file server using the telephone network for sending requests, and a broadcast medium (e.g. satellite transmission or data transmissions during the Vertical Blanking Interval on a regular TV broadcast channel [14]) for the downlink. In this case, each receiver sees an effective value of D proportional to the time between its request and the time the sender transmits the last packet. This tells us that having receivers partially overlapped and spread over large time intervals is actually beneficial for the protocol since it permits reception to complete without uplink communication even in presence of large losses.

A final consideration can be done on Fig. 6, which influences the parameters of the encoder. We have done our analysis under the assumption that $n \gg k$, so that it is highly unlikely that a receiver sees the same packet more than once (thus producing duplicates useless for decoding purposes). The graph tells us that (for losses up to 20%) it is extremely unlikely that a receiver will need more than twice the minimum number of packets to complete reception. So we can use this information to limit the value of n to be used in the encoder. It is important that n does not become too large because it can affect the performance of the encoder/decoder (e.g. for implementation reasons, using $n > 255$ with our encoder causes a slowdown of a factor of 2 or more). But the use of a small value for n also opens the possibility of computing the redundant packets only once and storing them on disk, to be reused subsequently.

V. Implementation

The protocol described in the previous section has been originally implemented for use on the MBone. The server is event driven, and is able to handle the transfer of several files simultaneously. It accepts requests on an UDP port, and returns the session key to the client using UDP, either on a unicast or a multicast address. The requested data is transmitted, according to the request, either using multicast UDP, or to a unicast UDP address. The latter feature can be used to exploit the loss tolerance of the protocol when used over a lossy network connection.

Since we use a connectionless protocol, wireless communication devices can be easily integrated in the system by simply receiving UDP packets and retransmitting them in each cell. A roaming user will be able to continue reception transparently while moving, without the need for any specific support for handoff. In fact, the basic services offered by the MBone will cause data to be broadcast to all cells where there are active receivers or beacons, and the packets lost while crossing cell boundaries will be easily recovered thanks to the FEC protection.

The default values of k and n have been set to 32 and 255 respectively, which provide a good compromise between speed and efficiency. The default bandwidth is 64Kbit/s; the maximum bandwidth is limited to 1Mbit/s, which is reasonably high to cover all practical needs[5] even on an Ethernet LAN.

Decoding costs can be of some concern at the receivers, since the latter might be resource-limited machines. By design, RMDP makes it possible for a receiver to determine if reception is complete or not without doing the actual decoding, but just looking at the identity of the various packets. So, even if a receiver is unable to complete reception in real time,

[5]The protocol is more useful when transfer times are long, since more clients are likely to participate. The bandwidth limitation comes from the need to avoid monopolizing the network for long intervals.

it can still store received packets and do the decoding offline, or at a slower speed. This permits to minimize the time the communication device – a power hungry subsystem – must be kept active. It is also noticeable that the robustness of the protocol to missing packets can be used to recover from losses caused by temporary inability to receive because the machine is busy doing the decoding.

In certain applications, the decoding costs can be further reduced by remembering that we use a systematic code, and data packets are sent in clear at the beginning. When transmissions are scheduled autonomously by the sender, many receivers will automatically synchronize with the beginning of the transmission, and will be able to collect a large number of packets in clear, thus needing to run the recovery process only for those packets (hopefully much fewer than k per slice) which have been lost. In this case the decoding cost at the receiver are only proportional to the number of missing source packets per slice, which means that decoding can be carried on at a much faster rate.

VI. Conclusions

We have presented a Reliable Multicast data Distribution Protocol (RMDP) which, unlike other reliable multicast protocols, is based on the use of software FEC techniques. Thanks to the approach used, the protocol is extremely simple to implement, makes an efficient use of both the downlink and the uplink, and is adaptable to a number of applications, from LAN-based to wireless/mobile ones (e.g. circular distribution of data, WEB browsing, access to databases in airports, stations, museums, etc.). Despite a software implementation of FEC is somewhat expensive, our results show that its performance is adequate for a wide range of applications, both in wired and wireless networks.

VII. Acknowledgements

This work has been partly supported by the Ministero dell' Universita' e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications", and by the Commission of European Communities, Esprit LTR Projects 20422 – MOBYDICK and 21926 – HIPPARCH (High Performance Protocol Architectures).

The author want to thank the anonymous reviewers whose suggestions greatly contributed to improve the clarity of this paper.

References

[1] M.Ammar, J.Wong, "The design of teletext broadcast cycles", Perf. Eval. vol. 5(4), pp. 235-242 (1985).

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

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

[4] J.W.Byers, M.Luby, M.Mitzenmacher, A.Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", TR-98-005, U.C. Berkeley.

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

[6] 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
`ftp://ftp.ee.lbl.gov/papers/srm_sigcomm.ps.Z`

[7] J.C Henrion, "An Efficient Software Implementation of a FEC Code", Proc. of IDMS'97, Darmstadt, Germany, Sept.1997.

[8] 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.

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

[10] M.Luby, M.Mitzenmacher, A.Shokrollahi, "Practical Loss-Resilient Codes", in Proc. of the 29th ACM Symp. on Theory of Computing, 1997.

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

[12] J. Nonnenmacher, E.W.Biersack, D.Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission", SIGCOMM'97, Cannes, France, 14-18 Sep.1997.

[13] J. Nonnenmacher, E.W.Biersack, "Optimal Multicast Feedback", Proc. of INFOCOM'98, S.Francisco, Mar.29-Apr.2 1998, IEEE.

[14] R.Panbaker, "The Transmission of IP over the Vertical Blanking Interval of a Television Signal", Internet Draft `draft-panbaker-ip-vbi-01.txt`, March 1997.

[15] L. Rizzo, "Effective erasure codes for reliable computer communication protocols", ACM Computer Communication Review, Vol.27, n.2, April 1997, pp.24-36.
`http://www.iet.unipi.it/~luigi/fec.ps`
Source code at `http://www.iet.unipi.it/~luigi/vdm.tgz`

[16] L. Rizzo, L.Vicisano, "A Reliable Multicast data Distribution Protocol based on software FEC techniques', Proceedings of The Fourth IEEE Workshop on High Performance Communication Systems (HPCS'97), 23-25 June 1997, Chalkidiki, Greece, IEEE.

[17] E.Schooler, J.Gemmel, "Using Multicast FEC to Solve the Midnight Madness Problem", Microsoft Research Tech. Report MSR-TR-97-25.

[18] 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.

[19] L.Vicisano, L.Rizzo, J.Crowcroft, "TCP-like congestion control for layered multicast data transfer", Proc. of INFOCOM'98, S.Francisco, Mar.29-Apr.2 1998, IEEE.

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