A Reliable Multicast data Distribution Protocol based on software FEC techniques

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

Lorenzo Vicisano†
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
email: l.vicisano@cs.ucl.ac.uk

Abstract

Applications requiring the reliable distribution of data to groups of clients would be supported perfectly by reliable multicast protocols. In many cases, the problem of congestion control (a major research issue otherwise) does not exist because downlink bandwidth is "owned" or can be preallocated to a particular server by independent means, but the problems of insuring reliable data delivery to large groups, and adaptability to heterogeneous clients, still remain.

These problems can be solved at once with the use of FEC techniques. In this paper we show the design of a Reliable Multicast data Distribution Protocol (RMDP) that we have built using these techniques, and discuss the implementation tradeoffs. Experimental results show that, albeit somewhat expensive, doing encoding/decoding in software is affordable for speeds in the Mbit/s range even on low-end PCs. Slower machines can still receive at high speed, thus optimizing network usage, by taking advantage of the fact that decoding needs not to be done in real time. Finally, our RMDP can work even without any feedback from the receivers, thus making it well suited to mobile/wireless systems.

1 Introduction

From the very early days, computer networks have been built around multicast/broadcast media. The exploitation of the multicast nature of the media has been hindered by the difficulty of extending this feature to wide area networks, and by the problem of building efficient and scalable reliable multicast protocols. In recent years some of these limitations have been overcome: networks and computers have become faster, multicast support has been deployed on the Internet [2, 3], and, especially, with the advent of multimedia some applications have been devised (e.g. audio and video conferencing) which do not require reliable data transfer. Many 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 reliable multicast algorithms which are scalable to large groups of heterogeneous clients and suited to a wide range of loss rates.

Unlike many previous works on this subject [6, 9, 16], our approach is based on a redundant encoding of data which makes the protocol tolerant to packet losses and client heterogeneity: since each received packet conveys information on a (potentially large) number of source data packets, reception of specific packets is not necessary; rather, it is the total number of received packets which determines the successful completion of the data transfer. This approach has large implications on the structure of the communication protocol: the need of feedback from the receivers is drastically reduced, with significant advantages on the complexity and scalability of the protocol and on its applications.

* Supported in part by the Commission of European Communities, Esprit LTR Project 20422 – "MOBYDICK", and in part by the Ministero dell'Università e della Ricerca Scientifica e Tecnologica of Italy.

† Supported by the Commission of European Communities, Esprit LTR Project 21926 – "HIPPARCH".

The paper is structured as follows. Section 2 briefly describes the problem of reliable multicast for large and heterogeneous groups. This Section includes a brief introduction to the principle of operation of the Erasure Codes which have been used to implement the protocol. Section 3 presents the principle of operation of RMDP. In Section 4 the various issues and tradeoffs related to the protocol's implementation are discussed, and performance data presented. Section 5 describes the current, publicly available, implementation of RMDP. Finally, Section 6 briefly discusses future developments of RMDP.

2 Reliable Multicast

Reliable Multicast (RM) protocols implement the reliable delivery of data to multiple receivers on top of the services offered by a communication network. There are different ways this goal can be accomplished, depending on the order source data packets are delivered, to each receiver and to the group as a whole. Totally Ordered, Causal, or Concurrent RM define three such possibilities, in decreasing order of complexity and guarantees offered.

The protocol presented in this paper implements Concurrent RM, which refers to the eventual delivery, to all receivers, of all data packets with no specific ordering constraints. Concurrent RM has a wide range of applications, since in many cases the ordering of messages locally to each receiver is not important in the first place, or it can be implemented very simply by tagging packets with sequence numbers. Total and causal ordering can be implemented at a higher level in the protocol stack, if necessary, by making use of Application Level Framing (ALF) principles [4].

The network may introduce duplications and reorderings of packets, originating from routing problems. Also, packet losses are unavoidable, because of noise and communication errors, even when the network preallocates resources for the connection. Congestion-related losses also come into play when a best-effort network is used; in this case, an even larger amount of losses is to be expected when multiple competing flows contend for the available resources.

Dealing with duplicates and reordering is relatively simple, and in case of Concurrent RM it is done above the transport layer: it just requires to assign sequence numbers to packets at the sender, and let the receivers handle the proper delivery of data to the upper protocol layers. Since each receiver can operate independently, there are no scalability problems related to network reorderings and duplicates. Handling packet losses is harder, though, because it requires the missing pieces of information to be transmitted more than once. Scalability problems might then arise in presence of multiple receivers, since each receiver might experience different losses and thus require different different packets to be resent. Commonly used techniques to recover from losses are based on ARQ (Automatic Retransmission reQuest) and FEC (Forward Error Correction), or combinations thereof.

2.1 ARQ

ARQ is based on the retransmission of missing data packets, which is triggered by explicit or implicit (based on timeouts) ACK/NACKs from the receiver(s). Multiple retransmissions can be necessary to insure delivery of all data packets to a receiver, but on average the number of retransmissions for each packet is $1/(1-p)$ where p is the packet loss rate (PLR).

ARQ is generally used in unicast protocols, because it is very effective, simple to implement, and it does not require any processing of the data stream. The are a few drawbacks in ARQ-based protocols, namely the need for a feedback channel (whose use can be minimized though) and the time (a full round-trip-time (RTT) as a minimum) required to recover missing packets. The latter is an important limitation when propagation delays are very large (e.g. in the case of fast satellite links or deep space communications), or when there are real-time applications (e.g. full duplex voice/video conferencing and control system).

ARQ scales badly to multicast protocols and large groups, because the chance of uncorrelated losses grows with the size of the group, and that might require the retransmission of the majority of packets [5, 7, 11, 15] even in presence of moderate PLR. This phenomenon is tolerable only in a few cases, e.g. when the group is small, or receivers have similar features and loss patterns. In all other cases ARQ performs poorly, since the aggregate PLR can become very close to 1. Another problem with ARQ in multicast protocols is that it requires precise feedback from the receivers to decide which packets to retransmit; depending on

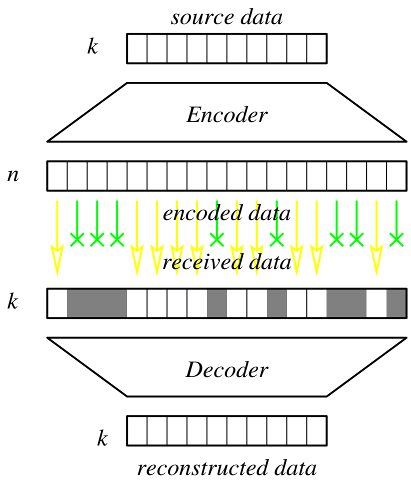

Figure 1: A graphical representation of the encoding/decoding process

the way feedback is propagated to the source, this might lead to scalability problems, and is unpractical in an environment where uplink communication is expensive.

2.2 FEC

FEC operates on a different principle, namely it anticipates some amount of losses, and obviates by sending redundant data which allow the receiver to reconstruct up to a certain number of missing packets. The communication process thus includes an *encoding* phase at the sender, where redundant packets are constructed from the source data, and a *decoding* phase at the receiver, where source data are extracted, if possible, from the available packets. A graphical representation of the encoding/decoding process is given in Figure 1. Because of the redundant data being transmitted, the PLR after decoding – i.e. the probability that some data packet cannot be reconstructed – can be made much lower than the raw PLR on the communication channel. FEC by itself does not guarantee reliable delivery (unless the number of redundant packets goes to infinity), but, by choosing a suitable encoding, the residual PLR at the receiver can be made arbitrarily small (obviously at a cost in the total transmission time), and those residual cases can be dealt with by using an ARQ-based protocol, if necessary at all.

FEC can be computationally expensive, since the entire data stream must be processed by the encoder, so that each transmitted packet carries information on a (possibly large) number of source data packets. Decoding can be expensive as well, depending on the encoding technique being used and the actual amount of losses experienced. This renders the technique unattractive for unicast protocols. In reliable multicast protocols, though, the advantages of FEC may overcome the encoding/decoding overheads. In fact, to some extent, the presence of the decoding process decouples receivers from each other. Independent losses do not combine so badly anymore, because the same redundant data allows different receivers to reconstruct distinct missing packets. The effect of independent losses becomes less important also because of the presence of fewer data units (groups of k packets as opposed to single packets) and of a much lower PLR after decoding. These combined events allow FEC-based multicast protocols to scale to much larger groups than plain ARQ-based protocols.

2.3 Erasure Codes

In network protocols, FEC can be implemented using Erasure Codes, which are a subset of Error Control Codes [1], largely used in the telecommunication field. Basically, an (n, k) Block Erasure Code 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. A code is called *systematic* when the encoded packets include the source packets in clear (systematic codes are much cheaper to decode when only a few erasures are expected). A block code is *linear* if the encoded packets are derived from the source packets by using linear transforms, which can be represented in matrix form as

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

where $\underline{x}$ is the source data (a vector of size k), $\underline{y}$ is the encoded data (a vector of size n) and G is an $n \times k$ matrix called the encoding matrix. G must be structured in such a way that any subset of k of its rows has full rank, so that the original data can be reconstructed by using any k components from $\underline{y}$.

In some cases, erasure codes are trivial to build, e.g. in a $(k+1, k)$ systematic code, G is the identity matrix followed by a row vector of all ones, and the redundant data is just the sum of the source data elements. For generic n and k, erasure codes can be constructed and studied basing on the properties of linear algebra. A complete discussion of the subject can be found in [1], while [8, 12] give a simplified description more closely related to erasure codes and networking protocols.

If computation are performed using ordinary integers, the components of $\underline{y}$ generally require more bits to be represented than those of $\underline{x}$. This expansion of data elements is very unpractical because it increases the size of data to be transmitted, and it may also create alignment problems which render subsequent processing much slower. This limitation can be overcome by performing computations in a finite field. A finite field [1] is a finite mathematical structure with two operations defined on its members – addition and multiplications – which is closed under both addition and multiplication (and their inverse). Most of the properties of linear algebra hold in finite fields, and the closure property is especially important since it allows operands and results to be represented with the same number of bits. While operations on a finite field are not directly supported in most processor, they can be performed at a relatively low cost even in software as demonstrated in [12].

In our implementation of RMDP we have adopted the (n, k) erasure code described in [12], which uses Vandermonde matrices with computations done over $GF(2^8)$. The code is based on the property that a polynomial of degree $k-1$ is fully specified by knowing its value in k different points. We interpret our data items as coefficients of a polynomial of degree $k-1$, and build the redundant information by computing the polynomial in $n-k$ different points, where the matrix representing the encoding process is known as a Vandermonde matrix.

3 RMDP

In this section we present the design of a Reliable Multicast data Distribution Protocol (RMDP) implementing Concurrent RM and able to transfer data to a potentially large number of heterogeneous clients. We assume that the source data has a known, finite size and is split into L packets of size s. Clients may have different receive speeds, experience different loss rates, and possibly start receiving at different times. Our protocol assumes a multicast downlink (from server to clients), is usable on asymmetric links, and makes a limited usage of the uplink when the latter is expensive (both in terms of monetary costs and of power, e.g. in the case of mobile, battery powered equipment). We fulfill all the above requirements by making use of FEC techniques.

The principle of operation of RMDP is very simple. Source data are encoded with high redundancy, using an (n, k) code with $n \gg k$. Data transmission starts at the request of the first client, and continues with a fixed rate (possibly repeating each packet more than once) until there are active clients. Ideally $k = L$, the data size measured in packets. Provided the same packet is not received twice by the same client (which is guaranteed to be true over a reasonably long interval if $n \gg k$), and because of the properties of erasure codes, it does not matter which ones, but only how many packets are received. Fast clients, able to keep up with the transmission rate, will complete reception after the first k packets; occasional packet losses can be recovered by simply accumulating more packets. Slow clients might be unable to receive packets at full speed, either because of their own limitations, or because they are connected over a low speed link. They will experience a high loss rate but will still be able to accumulate a sufficient number of independent packets given a sufficient reception time. If a client is exceedingly slow, it might receive the same packet twice, but this only translates in a slightly suboptimal reception time.

The basic protocol is structured as follows:

1. a client contacts the server requesting a file, and specifies the preferred parameters (e.g. data rate, block size) for the transfer.

System	CPU/Speed	c_d (MB/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: Operating speed of the decoding process on various architectures

2. the server replies with a set of parameters to be used to fetch the requested document. These include a *session key*, the address used to send the data packets, the file size, and other information of use for the client.

3. the server transmits a sufficient number of packets to allow the reception of all data even in presence of a moderate amount of losses.

4. in case the client experiences a halt in the data flow before the reception completes, a *continuation* request is sent to the server asking for more packets.

The first two phases of the protocol can be repeated if a request gets lost, since both actions are idempotent. The server does not retain state about the active clients, except from determining how many packets need to be sent and which bandwidth to use. The last phase is in many cases unnecessary, either because the initial amount of redundancy is sufficient to recover from losses, or because partially overlapped requests actually cause the transmission to last longer (see Figure 4). RMDP can be classified as a hybrid FEC-NACK protocol, although, in a proper multicast setting with many clients, phase 4 is rarely necessary even in presence of a large amount of losses.

4 Practical issues

The previous description only gives the principle of operation of RMDP. Many parameters were left unspecified, and will be discussed in this Section. Depending on the actual values of these parameters, the protocol can be tuned to different environments and applications. In this paper we focus our interest on a basic version of the protocol, meant to be used over multicast-IP on wide-area networks, for the transfer of medium-sized documents without any realtime requirement. Many applications fall within this category, including Web caches preloading, distribution of data such software packages or electronic newspaper, or even (to some extent) Web browsing. We do not impose any restriction on the number and type of receivers, although we expect them to be heterogeneous, have different receive bandwidths and experience different loss rates.

4.1 Encoding block sizes

In a practical implementation of our technique, k cannot be made exceedingly large. A large k improves the robustness of the code (in terms of erasure-recovery capability), but it also increases the encoding and decoding costs. With the encoding we have used, the encoding/decoding speed can be modeled as c_e/k and c_d/k respectively, where the constants c_e and c_d are very similar, and vary depending on the speed of the system. Table 1 gives the values of c_d on different systems, for a packet size of 1 KB.

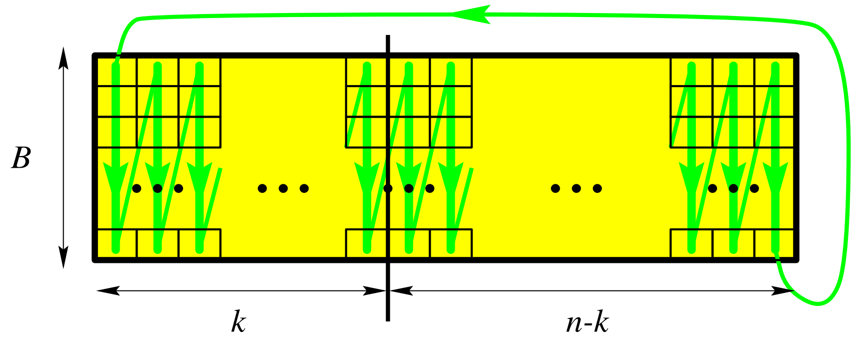

Figure 2: The arrangement of data and the transmission order at the server

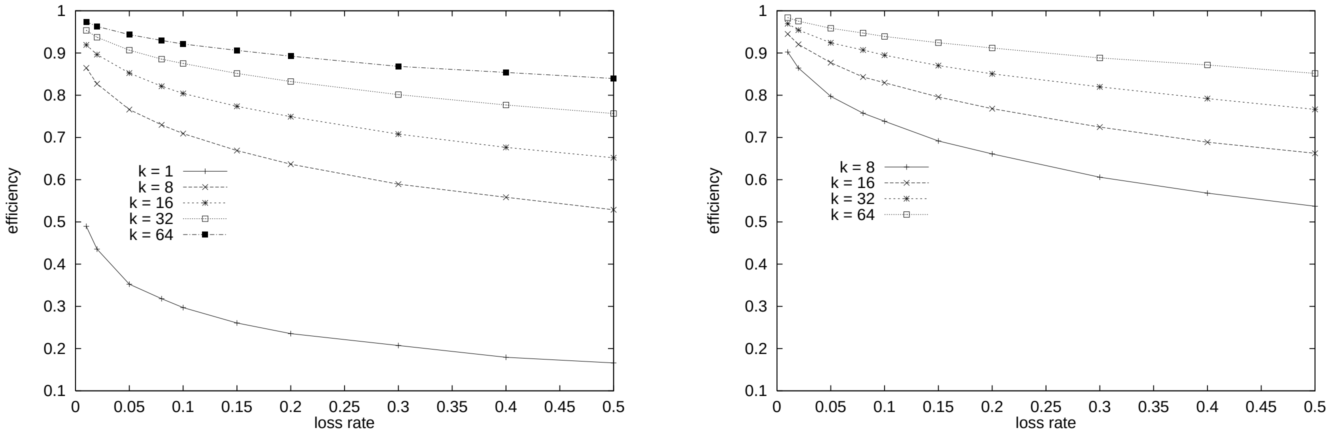

Figure 3: The efficiency of the protocol for random and burst losses, for a 2048-packets file and different values for k

These results should be considered as an upper bound for the cost of FEC encoding/decoding in FEC-based protocols: up to 2x speedups can be achieved by writing optimized implementations of the FEC code (our timings are derived from a general purpose C implementation). Also, in many cases the cost of encoding/decoding is much lower because the parties can rely on a large number of source (non-redundant) packets. Even in absence of such optimizations, our performance data show that speeds in the Mbit/s range can be easily achieved with current personal computers and values for k in the 30..64 range.

Basing on the above results, encoding/decoding overheads can be reduced to an acceptable value by splitting the source data into B groups of k packets each, where the value of k is fixed and $B = \lceil L/k \rceil$. On each group, an (n, k) code is applied to build the encoded packets; finally, groups are interleaved by transmitting packets in sequence, one per group, as shown in Figure 2. Such an arrangement makes the encoding/decoding costs independent of the file size, in exchange for slightly more restrictive conditions on received packets: not only at least L different packets must be received, but they must include at least k packets per group. Also, the effectiveness of the encoding technique (in terms of robustness to losses) is somewhat reduced, but not dramatically.

Figure 3 shows the efficiency of the protocol for different values of k, for a file of $L = 2048$ packets, in presence of random losses (left) and burst losses (right). Burst losses are obtained by means of a discrete time Markov chain with two states (A, B). A packet is lost when the system is in state 'B', while it is successfully delivered if the system state is 'A'. State transitions can happen with a given probability on each packet arrival. State transition probabilities are computed from the chosen average gap length (number of contiguous packet lost) and from the average loss rate. In Figure 3 (right) we have used a (quite short) average gap length of 5 packets, and the resulting protocol efficiency is not much better than the one for random losses. The use of a larger gap lengths shows that longer bursts turn into higher efficiency (see also Figure 5).

Although the efficiency increases with k (becoming 1 for $k = L$), it should be noted that the main source of inefficiency lies in the (almost complete) lack of feedback from the clients, not in the use of a fixed value for k. This appears clearly from the lowest curve in Figure 3 (left), which refers to the case $k = 1$ (corresponding to pure retransmissions, such as in ARQ protocols in absence of precise feedback). On the other hand, it should be kept in mind that RMDP is meant for bulk data transfer to many clients, so the additional transmissions are not wasted since they will be probably useful to many other clients, and the receive time is not a major concern, provided efficiency does not drop too much.

4.2 Initial setup

In the current implementation of RMDP, the client has the option of using unicast or multicast IP (actually, UDP) to send the initial request to the server. The reply from the server is always sent using unicast UDP.

There is a reason for sending requests using multicast. If the multicast request succeeds, then there is knowledge that there is multicast connectivity to the server, and the data transfer has a chance to proceed. This is important since multicast is not widely supported in the Internet yet, and some sites might be reachable via unicast but not via multicast. Second, this eases the scalability of the protocol in a context where the client is not interested in reaching a particular server, but only in having the requested file delivered. Any server listening to the multicast group used for sending the request can reply, making the service scale better.

Unicast requests can be used if the client wants to contact a particular server, or it wants to use the enhanced loss resilience of RMDP for unicast file transfer.

4.3 Sending rate

Deciding what transmission rate to use is critical in multicast-based applications, and this decision is strictly related to flow and congestion-control decisions. Currently, RMDP does not implement any form of congestion control, although we are actively working on this subject. In order to make the application network friendly, we have placed a hard limit on the maximum transmit rate for the RMDP server, which can be configured at server's start time. Reasonable values are between 64K and 1Mbits/s. However, data transmission does not always occur at the maximum rate. Clients can in fact specify their preferred bandwidth for receiving data, and the server tries to adapt to requests from different clients.

The adaptation of the sending rate to client's requests can be implemented by storing (at the server) a profile of the sending rate, as shown in Figure 4. A new request causes the sending rate to grow from 0 to the requested rate r_1 at request time, and back to 0 after a time $T_1 = (1 + p)L/r_1$ sufficient to transmit the whole document assuming a loss rate of p. Each subsequent request will add at most one corner point to the profile of the sending rate, with a processing overhead proportional to the number of total corner points. The amount of state (and processing overhead) in managing the transmission profile can be limited by allowing only a small number of different rates (so that requests tend to merge), and by adapting transmission shape so that new requests are handled without increasing the total number of corner points when they exceed a given limit.

If receive rates are too different, the slowest clients are likely to experience a relevant amount of losses. From the point of view of the client this would not be a problem if $k = L$, but since we use a fixed value for k in the encoder, we expect efficiency to become low for the client, as suggested by Figure 3. Furthermore, and more importantly, the network serving the slow clients would be flooded by useless data, preventing the correct operation of other network services. These two problems can be solved by sending data using multiple multicast channels. The simplest approach consists in grouping clients basing on the requested data rate, and using a different multicast channel for each rate. However this approach wastes bandwidth in case clients on the same subnet ask for the same data with different rates. A more efficient solution consists in using a layered approach, where the desired data rate can be achieved by listening to one or more channels. The number of different channels can be kept low by limiting the number of available data rates. A possible scheduling of packets on channels, which maximizes the efficiency of the protocol, is shown in [14].

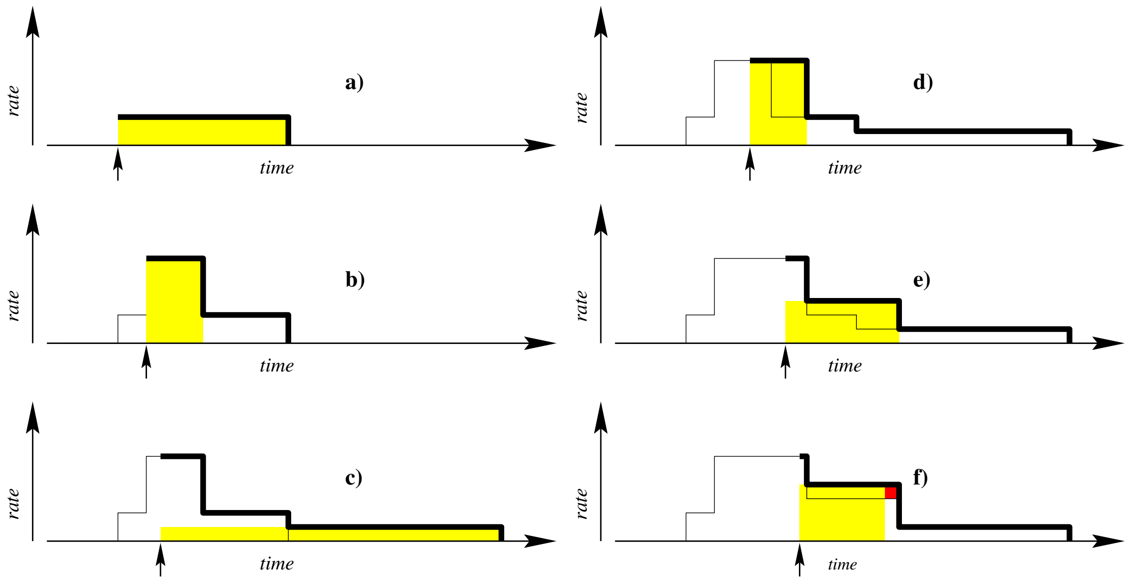

Figure 4: Management of the sending rate at the server. As requests, represented by arrows, arrive to the server, the existing transmission profile (thin line) is modified accordingly. The grey area represents the effect of the newly arrived request. In case the transmission profile becomes too jagged, corners can be removed with some minor reshaping (see the small dark area in the last picture).

4.4 Continuation requests

Limiting the amount of continuation requests from the client is important for two reasons. From the client's point of view, uplink bandwidth may be expensive, especially when using asymmetric channels or portable devices. From the server's point of view, many clients sending continuation requests could cause a feedback implosion which might saturate the network at the server, and in general cause the protocol to scale poorly. In RMDP, there is very little requirement for feedback from the clients, this being only necessary when the server stops transmitting and there are still clients which have not completed reception. By the time this happens, it is likely that very few clients are left, so the chance of a feedback storm is low. In order to further reduce the chance of such an event, the server indicates, in the header of each data packet, its willingness to accept continuation requests, and the number of packets still queued for transmission. Receivers can then determine if they have a reasonable chance to complete reception. If this does not happen, they schedule the transmission of a continuation request at a random time near the (expected) end of transmission. New clients joining the group, or continuation requests, cause the number of scheduled packets to increase, thus canceling pending continuation requests.

5 Implementation

An implementation of RMDP, containing most of the features discussed in the previous Section, is available from the authors [13], and has been used as a testbed for studying various issues related to reliable multicast protocols based on FEC. The code implements both a client, `rmdp`, and a server, `rmdpd`, is written in C and runs under several Unix-based systems. The `rmdp` client has a command line interface and uses a URL-like syntax to specify the requested files, e.g.

```
rmdp rmdp://some.server/some/file/name
```

fetches the desired file from the server specified in the URL. Command line options can be passed to specify the desired bandwidth, and the use of multicast or unicast for the transport of data.

The `rmdpd` server is run as ordinary daemon processes, and command line parameters can be used to specify the maximum bandwidth and the portion of the filesystem which is made available to clients. The server is event driven, and is able to handle the transfer of multiple files, using both multicast and unicast

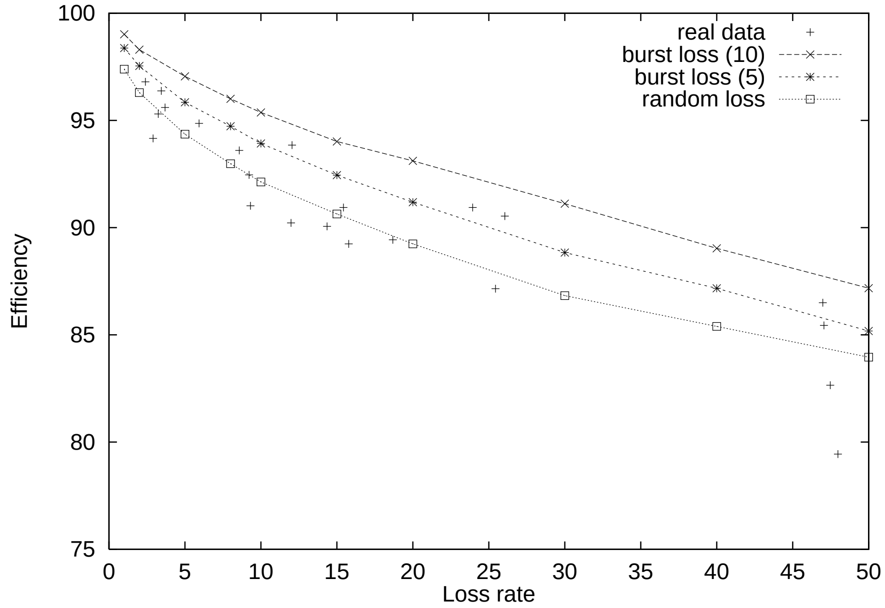

Figure 5: Efficiency of the protocol during real transfers, compared to simulation results for random and bursty losses with both 5 and 10 packet average gap length.

UDP as a transport. The latter feature can be used to exploit the loss tolerance of the protocol over a lossy network connection, or (with a simple frontend program) to interface the server to wireless communication devices.

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 default maximum bandwidth is set to 1Mbit/s, which is reasonably high to cover all practical needs even on an Ethernet LAN.

The initial experience with our prototype implementation of RMDP has been directed towards the evaluation of the actual efficiency in presence of losses. The protocol has been used satisfactorily to replace FTP on Internet paths where a loss rate up to 50% caused TCP-based applications to stall very frequently, or often fail because of excessive timeouts. Figure 5 shows the actual protocol efficiency for the transfer of a 2048-packets file, with 1 KB packets, and $k = 64$. The three curves in the graph represent the simulated behaviour, whereas the data points have been derived by from actual file transfers.

6 Conclusions and future work

We have presented a Reliable Multicast data Distribution Protocol (RMDP) which, unlike other similar 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 ones (e.g. workstation booting, file distribution) to wireless/mobile ones (e.g. circular distribution of data, browsing databases in airports, stations, museums). 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.

A prototype implementation of RMDP, including both client and server, is publicly available [13], and compiles on several Unix architectures. We are using this implementation both as a file-transfer tool, and as a testbed to run experiments on reliable multicast protocols and study some improvements for the protocol itself. RMDP is evolving along several lines. First, we are developing variants of the protocol which are better suited to different operating environments (optimized for transfer speed, bandwidth usage, or uplink usage). Second, there is ongoing work to add congestion control to the protocol, so as to make RMDP more network friendly. Third, the FEC encoding currently in use is not particularly efficient for extremely large files leading to large values of B. For there cases, we are investigating the use of different encoding techniques, such as the ones presented in [10]. These encoding algorithms are extremely fast for large values of k, but have some drawbacks when compared to the encoding we use.

References

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

[2] S.Deering, D.Estrin, D.Farinacci, V.Jacobson, C.-G.Liu, L.wei, "An Architecture for Wide-area Multicast Routing", ACM SIGCOMM 94, pp.126-135, UCL London, Aug. 1994

[3] S.Deering, "Multicast Routing in a Datagram Internetwork", PhD Thesis, Stanford Univ., Dec. 1991

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

[5] M.Handley, "An Examination of MBone Performance", ISI research report ISI/RR-97-450.

[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, available as `ftp://ftp.ee.lbl.gov/papers/srm_sigcomm.ps.Z`

[7] 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.

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

[9] J.C. Lin, S. Paul, "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`

[10] M.Luby, M.Mitzenmacher, A.Shokrollahi, D.Spielman, and V.Stemann. "Practical loss-resilient codes", Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, page 150, El Paso, Texas, 4-6 May 1997. To appear.

[11] 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`

[12] L.Rizzo, "Effective erasure codes for reliable computer communication protocols", ACM Computer Communication Review, April 1997. Also available as DEIT Technical Report LR-970115, `http://www.iet.unipi.it/~luigi/fec.html`

[13] L.Rizzo, L.Vicisano, `rmdp` source code, available at `http://www.iet.unipi.it/~luigi/rmdp.tgz`

[14] L.Vicisano, "Notes on a cumulative layered organization of data packets across multiple streams with different rates", available as `http://www.cs.ucl.ac.uk/staff/L.Vicisano/layers.ps`

[15] M.Yajnik, J.Kurose, D.Towsley, "Packet Loss Correlation in the MBone Multicast Network", IEEE Global Internet Conf., pp.94-99, London, Nov. 1996

[16] 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`