To appear in Proceedings of the Third International Workshop on High Performance Protocol Architectures HIPPARCH '97 Uppsala, Sweden, June 12-13, 1997

One to Many Reliable Bulk-Data Transfer in the MBone

Lorenzo Vicisano, Jon Crowcroft
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
email: {L.Vicisano,J.Crowcroft}@cs.ucl.ac.uk

Abstract

In this paper we depict and evaluate the performance of a protocol for reliable bulk-data transfer from one sender to many receivers simultaneously, using the Internet multicast infrastructure (MBone). The protocol, featuring a *TCP-friendly congestion control* algorithm, has been designed aiming at achieving a complete scalability of the system with the respect to the number of receivers. For this reason both reliability and congestion control are carried out by receivers, avoiding to involve the sender in the congestion control feedback loop and relieving it from the burden of carrying out retransmission for all receivers.

The receiver-driven congestion control algorithm presented is based on a redundant layered organisation of data, where redundancy is used to spread the information being sent into a large number of data units, providing receivers with flexibility in accepting some of them rather than others, and still being able to complete the reception. Reliability is based on a probabilistic approach and is achieved using redundancy and continuous transmission, providing graceful degradation of the protocol efficiency when the loss rate increases.

Other contributions in this paper include: simulation of the protocol in competition with TCP with drop-tail and RED routers; analysis of the stable operating point; implementation and measurements of the protocol (future).

1 Introduction

In pursuit of the design of a one-to-many reliable bulk-data transfer protocol that runs on top of the Internet multicast service, there are two main issues have to be faced: *reliability* and *congestion control*. As we wish to address scenarios comprising thousands of receivers, a fully scalable approach has been chosen. We handle reliability using FEC techniques, and congestion control by means of a receiver-driven scheme. For scalability reasons, we have also chosen not to define any explicit forms of group membership semantics: the sender is completely unaware of number and identity of receivers, all it have to know is how long to keep on transmitting a given file. That can be accomplished out of band, using feedback between high level management[1].

The implementation of the receiver-driven congestion control is based on a redundant layered organisation of the data being transmitted. In this case redundancy is used not to provide robustness with the respect to transmission error, but flexibility with the respect of usefulness in receiving a given packet. A redundant encoding could, in principle, allow us to

[1] One simple scheme is simply to elect some sample sites, and allot each a reporting time, scattering the report times across some period with a uniform random interval, to avoid implosion, in the manner of RTCP receiver reports[14]. When the application using the protocol is NNTP [10] or ICP[18], for News distribution, or web inter-cache protocol traffic, then the organisation of the elected control servers can easily be driven from the hierarchy already present in the news and web server organisation. Another simple protocol to accomplish this task is discussed in [13]

spread out the original information —e.g. to obtain n data packets from k original source packets— so that what counts is the amount of *data* received, not the actual information —i.e., dealing with packets, we can rebuild all the original k source packets once k packets are received out of the n sent, no matter which they are.

Given the redundant layered organisation of data, receivers are allowed to chose their receiving rate by means of joining the appropriate layers, performing this way the congestion control action independently by each others —although some coordination of nearby receivers might be needed to have a more effective action on shared network bottlenecks.

We note that the redundant encoding we have used, both to accomplish FEC and to provide receivers with flexibility, introduces a very small overhead in bandwidth usage, compared to a plain data transmission with ARQ-based selective retransmission for each receiver. This is due to the fact that a received packet is either entirely useful —accounting for its size to the reception progress— or completely useless, the former being by far the most likely case, and that the same packet can be used in different receivers for filling different 'gaps'. Moreover receivers are able to cut off the forwarding of packets as soon as they have collected all the necessary data, by means of the MBone pruning capability.

In the following we will describe with some more details the layered organisation of data and then we will present a receiver-driven congestion control scheme that provides fairness with respect to underlying TCP connection. We also address some related problems such as receiver coordination for congestion avoidance.

2 Layered Organisation of Data

We transmit data and redundancy packets over multiple, say n_c, *layers* (C_0—C_{n_c-1}), using a different multicast address for each layer. Receivers are allowed to chose their *subscription level* by joining/leaving multicast sessions. To obtain a more effective usage of the available bandwidth, subscription levels, or simply levels, are organised in a cumulative way: level L_{i+1} is obtained from level L_i by adding layers C_{i+1}, thus $L_i = \bigcup_{j=0}^{i} C_j$. Moreover, in order to bind the number of layers and still be able to have a large range of available data rates, we distribute exponentially the aggregate bandwidth of levels, so that level L_i has an aggregate bandwidth $bw(L_i) = b_0 2^i$. From this definitions follows that $bw(C_0) = b_0$, $bw(C_i) = b_0 2^{i-1}$. It is useful to define $R = 2^{n_c-1}$, the ratio between maximum and minimum bandwidths of subscription levels.

Receivers can control their receiving rate by choosing their subscription level. This way they establish a tradeoff between bandwidth usage and receiving time, similarly to what is sometimes done with layered multimedia streams [8], where quality can be traded to save bandwidth. In order to allow receivers to fully exploit the bandwidth they use, we must encode data in a flexible way and organise it properly across layers. That can be done using a suitable *block redundancy code* able to recover from erasures (see [12]). In redundancy techniques based on systematic[2] block redundancy codes, $n - k$ redundancy packets are computed from –and transmitted along– k source data packets; at the receiver side all the k original data packets can be recomputed provided that at least k *different* packets are received out of the n packets transmitted, no matter which packets are received. In our case, we can use this technique to obtain $n = Rk$ different packets from the original k data packets, this way, while we transmit k packets in layer C_0, we can transmit the remaining $(R-1)k$ packets in the other layers. That allows receivers to receive k *different* packets, no matter which level they are subscribed to.

A straightforward implementation of the technique described above implies encoding all at once the source data packets being transmitted. This provides the maximum flexibility but has the drawback of requiring a large encoding/decoding overhead, that grows with the values of k and n. For this reason we use a different solution that allows to choose a tradeoff between flexibility and coding overhead. We split the data to be sent into B *blocks*

[2] A code is called systematic when the encoded data include the source data in clear.

of fixed size. Each block j is made of k data packets, say $p(i,j), i = 0$—$(k-1)$. $n-k$ redundancy packets are computed in each block, say $r(i,j), i = 0$—$(n-k-1)$. As before we choose $n = Rk$, so that the total number of packets is R times the number of source packets. This time all the original data packets can be recomputed, provided that k different packets for each block are received. That impose the need to use a proper packet organisation across different layers, in order to minimise the number of useless packets received —if we already have k packets of a given block, each additional packet of that block is useless. [16] shows two possible packet organisation, that allow to obtain good bandwidth exploitation efficiency, interleaving blocks across layers. Some simulations performed in [16] show that the receiving efficiency —defined as the total number of packets received divided by the file length in packets— is always larger than 90% for common loss pattern, such as burst losses or random uncorrelated losses.

Concerning the redundant encoding algorithms, [12] shows a fast software implementation of such an algorithm, fast enough to allow on-line encoding with a throughput up to 1—$2Mb/s$.

The redundant packet encoding is exploited not only to have the flexibility we discussed, but also to cope with packet loss. As a matter of fact, the redundancy allows an effective recovery from packet erasure by means of the reception of additional packets beyond those that would suffice if there were no losses. This way of providing reliability is a probabilistic approach, that avoids retransmissions altogether, only requiring the sender to keep transmitting for the sufficient amount of time.[3][4] It also make this protocol suitable in scenarios where reception are started in a uncoordinated fashion at each site, but still statistic overlapping exists. [13] discusses further this issue, also showing graphs reporting the reception completion probability versus the transmission extension for various loss rate and patterns.

3 Congestion Control

In the receiver-driven congestion control scheme we propose, each receiver is in charge of controlling the data rate flowing toward it, in order to avoid congestion in the network. That is possible thanks to the pruning capability of the IP multicast routing, that cuts a subtree away from the distribution tree for a given pair (source, group) when all the receivers lying in that subtree leave the group.

In the following we describe the behaviour of our congestion avoidance scheme compared to the TCP congestion control algorithm ([6][15]). TCP congestion control is based on a closed feedback loop that involve both sender and receiver. It is performed by the sender by means of controlling the amount of data injected into the network. At all times the sender's congestion window (`cwnd`) limits the amount of data in transit [5]: after a `cwnd` worth of data has been sent, new data can be injected only if the receiver acknowledges the reception of old data (`ack`). TCP performs congestion avoidance halving `cwnd` whenever a congestion is detected in term of a packet loss [6]. Moreover TCP continuously probes for remaining bandwidth by increasing its congestion window linearly (one MTU each round trip time). Beside what is described above, a number of other mechanisms exist to recover from packet loss and to deal with transient phases.

While the TCP congestion control scheme involves both sender and receiver —the sender performing the actual control, and the receiver providing the sender with the needed feedback information– we have avoided including the sender in the control loop in our multicast protocol, for scalability reasons. Both congestion detection and avoidance are performed

[3] This also makes the protocol suitable for asymmetric paths, or paths with no return channel at all (e.g. satellite channels).

[4] We believe a deterministic completion protocol to be part of the overall membership management layer's functionality.

[5] The window actually used is the minimum between `cwnd` and the receiver advertised window (`awnd`).

[6] In turn, packet losses are inferred from duplicated acknowledgement or retransmission timer expiration.

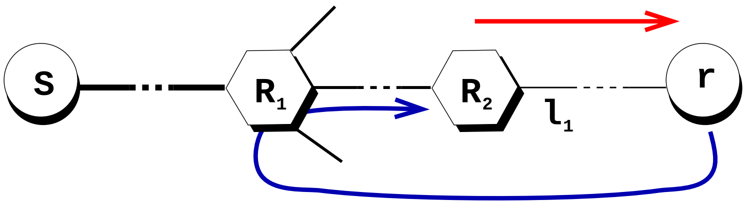

Figure 1: Congestion control feedback loop.

by receivers: the former checking for gaps in the packet sequence numbers, the latter joining/leaving multicast groups.

Referring to figure 1, S is the source of data, r is a given receiver, l_1 is the congested link (the one with lower available bandwidth along the path from S to r), R_1 is the router lying on the last branching point of the multicast distribution tree towards r before l_1, and R_2 the router feeding l_1, that drops packets when l_1 is congested; R_1 and R_2 can possibly coincide.

As can be see in figure, our feedback control loop is made up of R_1, R_2, l_1 and r. l_1 is the system being controlled, r/R_1 are the direct control chain —r adds/drops layers and R_1, as a consequence of that, resumes/stops forwarding of packets to r. The reverse (feedback) chain is R_2/r (R_2 drops packet when l_1 is congested, r detects the losses).

Note that the sender is not involved in the feedback loop, that allow us to scale this system up to a whatsoever large number of receivers.

Unfortunately, in an actual multicast transmission, things can be more complicated than those shown in figure. This is because more than one receiver can lie after the same bottleneck l_1. When that happens a number of loops exists, all of them involving R_1, causing several problems of interference and coordination. This issue will be discussed further in 3.1.[7]

Given that receivers are able to detect loss of packets and to regulate the throughput of the data flow, they can emulate the TCP behaviour as follows:

1. when a packet loss is detected, drop a layer, halving the data flow rate (multiplicative decrease);

2. after a time $t_p(i)$ is spent in the current layer, i, add a new layer, doubling the throughput (additive increase); $t_p(i)$ is proportional to the throughput associated with layer i.

Point 1) is a straightforward emulation of TCP congestion avoidance. Point 2) emulates TCP linear increase by probing for bandwidth left in the channel. In this phase, TCP increases its `cwnd` by one MTU for each Round Trip Time, this way `cwnd` gets doubled after $\texttt{cwnd} * RTT$, RTT being the time to transmit a `cwnd` worth of data. Since the throughput is $T = (\texttt{cwnd} * B)/RTT$ bytes per second, B being the MTU size in bytes, `cwnd` gets doubled after $(T * (RTT)^2)/B$ sec. This time is proportional to the throughput, as point 2) states.

The congestion control performed as above presents, however, a number of differences from that of TCP. First of all TCP congestion control is based on the 'conservation of packets' principle [6]: a new packet isn't put into the network until an old packet leaves; this allows robustness in the face of short-term load changes, in fact it makes the throughput decrease as the round trip delay increases. In our case we miss this feature being able to control the throughput only. However this is not a crucial point if load changes are symptom of incipient

[7] In other words, heterogeneity in the multicast delivery tree is caused both by different link speeds, and by the presence of other traffic, both unicast, and multicast, some or all of which may be operating congestion avoidance mechanisms such as TCP's our the one we are devising here. The dynamics of the situation make this a complex problem.

network congestion, as a matter of fact, in this case, the TCP conservation principle only has the effect of postponing the need to use the congestion avoidance algorithms.

Another important difference is that our protocol has a limited set of throughputs allowed (exponentially distributed), while TCP is able to adjust its throughput which much finer granularity. This has two major consequences: one is that we obtain a worse bandwidth utilisation (see 4), the other is that the minimum throughput allowed can higher than what the network can sustain; for this reason we must provide for a mechanism that aborts the data transfer if needed.

Finally the major disadvantage that we have with the respect to TCP is that multicast leave-group operation is slow to be performed, yielding to large reaction time to face congestions. As a matter of fact, most of the time, TCP is able to detect congestion within 1—2 RTT and to react in less than 1 RTT. We are able to detect congestion earlier [8] but the reaction involves leaving the multicast group corresponding to the upper layer, which takes 1—2 sec. with the current technology[9] This issue will be discussed further in 3.2.

3.1 Implicit Synchronisation

As noticed above, the uncoordinated behaviour of receivers sharing the same bottleneck presents several disadvantages, for example:

(a) the action of a receiver dropping a layer (leaving a group) has no effect unless all the receivers sharing the bottleneck drop that layer;

(b) if a receiver causes congestion on the bottleneck by adding a new layer, another receiver can interpret the losses as a consequence of its too high level of subscription and and drop a layer;

(c) if two receivers behind the same bottleneck have different subscription levels the bottleneck bandwidth allocated to that protocol instance is not fully exploited;

(d) as with *(b)*, but the case of multiple applications using separate instances of this protocol across the bottleneck, unaware of each other.

All above suggests that some form of coordination between receivers is needed.

Addressing point *(d)* requires coordination among different instances of a protocol unaware of each others. As this introduce several problems, most communication protocol do not address it. Nevertheless coordinating different instance of a given protocol that share the same bottleneck would apport several benefit, allowing more fairness and a better exploitation of the available bandwidth.

McCanne et al. in [8] proposed an explicit form of coordination among receivers, belonging to the same protocol instance. In their solution (*shared learning*) each receiver, before performing a *join-experiment* (add a layer) to infer the capacity of the links, takes care of informing the other receiver, specifying the current subscription level as well. This way, if the experiment is unsuccessful, all the receivers sharing the bottleneck will see the loss rate increased, and will avoid individual attempts. Unfortunately nothing can be inferred by receivers that wont see any increasing in the loss rate; as a matter of fact that can be due either to the fact that the experiment has been successful, or to the fact that they do not share any bottleneck with the receiver performing the experiment.

Some different explicit forms of coordination are possible. In theory some protocol could be used to aggregate all the receivers sharing a given bottleneck, perhaps exploiting the fact that all of them see the same loss pattern.

We are considering a scheme based on the actual loss pattern. Since packets carry a sequence number, each receiver sees a pattern over a sample period (which can easily be

[8] Likewise TCP, the receiver assume network congestion when it detects gaps in packet sequence numbers, but in this case the event does not need to be notified to the sender.

[9] Most of this time is spent to accomplish the IGMPv2 fast leave ([3]).

synchronised by protocol definition), of arrivals and erasures. This can be used to define a unique mask, which can be combined (e.g. xor-ed) with the base multicast group, to generate a new group, which could then represent a set of receivers who have experienced the exact same loss pattern. The problem with this approach is that it proliferates multicast address/groups/trees, since it generates a new group for each of all the occurring possible loss patterns. This has a high cost in multicast tree maintenance (both IGMP traffic, and router state), and in any case, some of the groups may be different where the difference in packet loss was due to true bit errors rather than congestive loss. We could reduce this proliferation two ways, both to make the aggregation of receivers more representative of the congestive loss patterns, and also to reduce load on the multicast infrastructure:

1. We could treat consecutive erasures as if they are *packet burst* losses. In "drop-tail" routers, we expect losses to occur in bursts, as traffic bursts/trains overload queues.

2. we can apply our "dynamic receiver regrouping" periodically, rather than continuously, and use long sequences of packet loss to find slow changing correlated regions.

An advantage of this approach is that it involves no explicit receiver coordination protocol. Another advantage is that it could be used to coordinate different instances of the same protocol (point *(d)* above), by means of using a fixed base address. The main problems are associated with a degree of uncertainty about how well one can actually detect loss correlation in the Mbone. Work by Handley (see [17]) suggests that it is easy to find some levels of correlated loss for some groups, but that there remains a hard-core of sub-trees of the network for which there is a lot of noise in the loss signal.

We have chosen to avoid any explicit coordination protocol because it could introduce scalability problems and, in same cases, could be slow to converge. As an alternative we propose to keep receivers sharing the same bottleneck implicitly synchronised, as far as possible.

Since we can assume that all the receivers behind a given bottleneck see the same losses and since they run the same algorithm, they should, in theory, be synchronised. Actually they are not synchronised for a number of reasons, including the fact that they do not see exactly the same losses —some of them could see some more packet losses due to noise on unshared links. This kind of errors tend to accumulate, making the the behaviour of receivers to deviate from each others. Another source of de-synchronisation is the fact that not all receivers start receiving at the same time, so that they have a different history.

To cope with that, and to prevent errors from accumulating, we have chosen to introduce *synchronisation points*. Synchronisation points are either explicitly sent by the sender or implicit in the data packet sequence numbers. They can also be packet loss events, or, more precisely, receptions of out of sequence packets. Receivers run the control algorithm basing on what they have seen since the last synchronisation points, and erase their memory concerning past history after each synchronisation point.

This way of acting has the advantage of preventing errors due to independent losses to accumulate, and allows receivers joining later to re-synchronise quickly. However, it has the drawback that the congestion control algorithm can only rely on a short time history. Moreover some sources of loss of synchronisation still remains. One of them is the fact that join and leave operation have different latencies for each receiver and that data packets propagate with different latencies. This is not, however, a serious problem because each receiver will eventually see a given event, react to it and, after that, experience the outcome of its action. As a matter of fact, if the reaction is a leave-group operation, it will have effect after all receivers have left the group, thus possibly introducing an additional delay before the outcome is perceived. If the reaction is a join, it will have effect after the first receiver performs it; anyway, even receivers with high latency will see its effect after the event that triggered it, thus preserving causality.

3.2 Sender-Initiated Probe

Feedback control loops characterised by a long delay path tend to be unstable. In some preliminary simulations, we actually observed an instability of our congestion control scheme, due to the presence of the long-delay leave. As a matter of fact, when a protocol instance, doubling its throughput, takes more than its fair share and causes congestion, a large number of packets get dropped before that instance succeeds in backing off. This cause all the other protocol instance sharing the bottleneck to back off as well, possibly halving their throughput more than once.

To cope with that we have devised two mechanisms: first we prevent a cascade of back offs happening by ignoring network congestion signal for a short time after a previous back off. This time is set to the leave delay plus a short additional time, to wait for queues to decongest.

As a second scheme, we probe for spare network capacity before increasing the subscription level, without actually subscribing the next level (joining the group). This is done by means of *sender-initiated probes*: periodically the sender doubles the bandwidth in each level by anticipating the transmission of some packets in all the layers. That has the net effect of doubling the transmission rate in each level. Moreover, during probes, data is transmitted in burst of two packets arranged back-to-back. In the case of Drop-Tail routers, this makes the dropping probability of probe-packets higher than that of normal packet.

3.3 The Congestion-Control Algorithm

On the sender side, packets of size B bytes are transmitted in all the layers. Transmission is properly phased in each layer so that the aggregate flows have constant packet inter-time. Flow associated to level L_i, for instance, presents a packet inter-time $t_i = t_0/2^i$, $t_0 = 1/b_0$ being the inter-time in layer C_0.

In each level L_i, every $W * 2^i$ packets, a burst of length $Y * 2^i$ is transmitted. The inter-burst time and burst length have the same duration in time in all levels —respectively $W * t_0$ and $Y * t_0/2$. A burst takes places in all levels at the same time, and is followed by a relaxation period of the same length as the burst. Bursts are made of a couple of packets transmitted back-to-back, couples being spaced the normal packet inter-time (t_i).

The receiver behaves as follows:

- if it is receiving a normal flow (non-burst) and *sees a loss*, **drop a layer** and setup a timer t_h to prevent further losses, possibly due to the same congestion, causing another layer to be dropped before the timer is expired;
- if it has received the last packet of a burst, and there have been no losses in this burst and has experienced *at least* $P * 2^i$ *bursts* since it lastly joined the current layer (i), **add a layer**.

Note that detection of a missing packet and end of a burst are synchronisation points, but this algorithm does not conform perfectly to the no-history principle because the 'current level of subscription', is not forgotten across synchronisation points, and the burst counter seen in the current level is not reset.

3.3.1 Parameter tuning

Various parameters (B, t_0, W, Y, t_h and P) must be tuned.

$W * t_0$, the inter-burst time, should be small to allow a quick rising time at the lowest levels. In turn W cannot be too small to allow to build the burst/non-burst pattern in L_0. Finally t_0 cannot be so small as to have a throughput small enough in L_0 without having B too small. The smaller acceptable values for W and Y are 8 and 2 respectively; using these values, in L_0, we have a two-packet burst and six-packet normal rate, with burst, relaxation and normal period lengths of t_0, t_0 and $6 * t_0$ respectively in all the levels, and $8 * t_0$ minimum rising time. Concerning t_0, to obtain a minimum rising time of 1—2 sec., it should be 250—125 ms; that allows minimum bandwidth b_0 of 1—2 KB/s with packet size $B = 256$ bytes (a smaller packet size is not convenient). t_h should be tailored to the leave delay; as a matter of fact, is very likely that a congestion will not be extinguished before the minimum reaction time (lower bounded by the leave delay), hence further reactions should be prevented for that amount of time.

The last parameter to set, P, allows us to tune our protocol in order to make it TCP-friendly ([4]). In [4], Floyd and Fall showed that, basing on a simple steady-state model, it is possible to compute an upper bound of the bandwidth a that TCP connection uses, as a function of the loss rate (p) it experiences. They found the following relation:

$$T_{TCP} \leq \frac{\sqrt{3/2} * B_{TCP}}{RTT * \sqrt{p}} \tag{1}$$

where T is the throughput of the connection and p is the loss rate. A relation identical to 1, with slightly different constant, has been found in [11] using a more rigorous model. Using a simplified model of our algorithm, we also found the relation between throughput and loss rate in our protocol. The model assume that receivers oscillate between two adjacent subscription levels. With this model we found the following approximation for the throughput:

$$T \simeq \frac{B}{t_0 * \sqrt{PW} * \sqrt{p}} \tag{2}$$

The equation 2 above is derived in A, where an upper and lower bound for the throughput are computed as well. In this formula, p is the burst-error rate, computed considering closely spaced losses as a single loss. As a matter of fact our congestion control algorithm considers losses closely adjacent in time as a single congestion signal; that is also true in most of the more recent implementation of TCP.

If we want our protocol to be as aggressive as TCP, it must be $T = T_{TCP}$, which gives

$$P = \frac{2}{3W} \left(\frac{B * RTT}{B_{TCP} * t_0} \right)^2 \tag{3}$$

P plays a role similar to RTT in TCP: the smaller is P the more aggressive is the protocol. The smallest value of P is 1 (P must be positive integer). With the values of W and B established above (8 and 256 respectively), our protocol can be at most as aggressive as a TCP connection with about 430 ms RTT and 256 byte MTU size. Hence it tends to be somewhat less aggressive than an average TCP connection. In the next section we will see, however, that the comparison of these two approximation for the throughput is not so straightforward; in fact p assumes slightly different meaning in the two formulas, because our protocol tends to aggregate losses more than TCP does.

4 Performance Evaluation

The protocol described in this work has been implemented and functionality-tested. Performance evaluation, however, has been carried out, for the time being, by means of a simulation. For this purpose we have implemented the protocol in the `ns` network simulator [9], partly in C++ and partly in otcl [19]. Both the real protocol implementation and the simulation code are available from the authors.

For all the experiments presented in this section we have used the simulated network topologies of Figure 2. The simulations have been run using both Drop-Tail routers and RED routers. All routers have been used with the default `ns` configuration, i.e. queue size of 50 packets, RED thresholds of 5 and 15 packets and queue weight of 0.002.

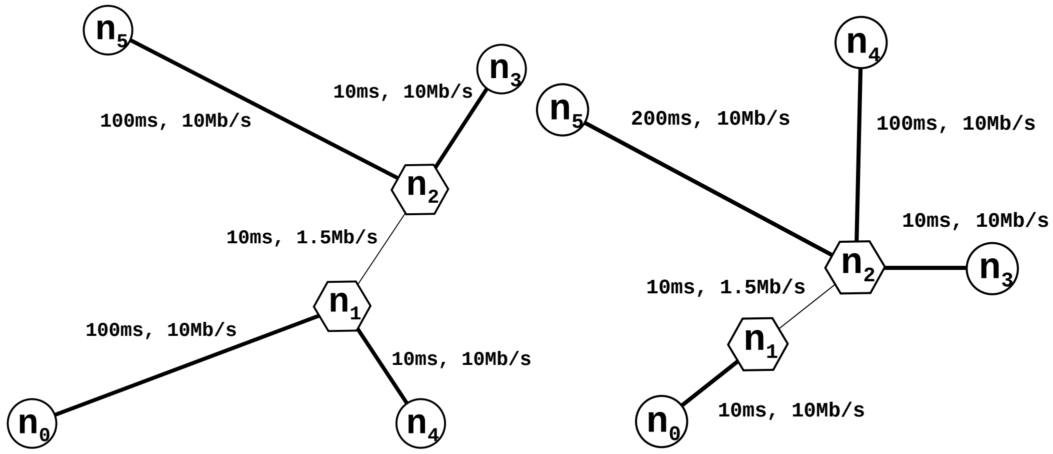

Figure 2: Network topologies used for experiments, left: a, right: b. Numbers close to each link are latency (ms) and bandwidth (Mb/s) respectively.

As far as our congestion control algorithm is concerned, we have used the following parameters: $B = 256$ bytes, $t_0 = .125$s (hence $b_0 = 2$KB/s), $W = 8$ and $Y = 2$ and $P = 1$. All the simulation has been run using a leave delay time of 1s, and we have set t_h 1.1 times the leave delay.

In the following, for comparison purposes, we will use the New-Reno variant of TCP described in [2] and implemented in `ns` by the authors of [2] according to suggestions from [5]. New-Reno TCP is a slightly modified version of Reno-TCP [7] that includes all Reno and Thaoe features (Slow-Start, Congestion Avoidance, Fast Retransmit and Fast Recovery) and fixes some problems of Reno's Fast Recovery preventing from the 'pipe' from emptying when multiple packets belonging to the same `cwnd` are lost.

The first experiment we run is to evaluate whether various protocol instances, competing for the bandwidth of a bottleneck, are able to achieve fairness among themselves. We spread couples of sender/receiver on the two paths with different delay n_0—n_5 and n_4—n_3. Each receiver was started at a random time in the interval $[1, 1 + N/2]$s, N being the number receivers.

Table 3 shows some parameters measured using 2, 4, 8, 16 and 32 protocol instances. The first parameter shown is the fairness index [1] that is defined as:

$$fi = \frac{\left(\sum_{x=0}^{N-1} T(x)\right)^2}{N \sum_{x=0}^{N-1} T(x)^2}$$

where $T(x)$ is the throughput of the x-th protocol instance. fi always lies between $1/N$ and 1: when all the instances get a perfectly equal share, fi is equal to 1; when one of the instances acquire all the available bandwidth, fi is equal to $1/N$. From table 3 we can see that our protocol achieves a good fairness both with RED routers and with Drop-Tail routers, although, in the last case, we measured an appreciably worse fairness with 4 and 8 protocol instances.

Table 3 also shows three different loss rate indexes: the loss rate seen at the router for that instance (the drop rate), the one experienced by receivers and the burst drop rate. In the bursts drop rate, closely spaced losses (namely less than .5s) are considered as a single loss. The difference between the values of the first two loss rates is due to the fact that receivers, once they have left a layer, do not take into account loss in it any more, while packets still reach the router until the leave takes effect. The third parameter (burst loss

protocol instances	2	4	8	16	32
RED					
fairness index	0.998	0.978	0.992	0.993	0.994
router loss rate	0.0054	0.0125	0.0250	0.0354	0.0737
perceived loss rate	0.0030	0.0070	0.0143	0.0231	0.0495
burst loss rate	0.000077	0.000510	0.001752	0.007092	0.021459
bandwidth exploitation	0.563	0.624	0.696	0.776	0.907
Drop-Tail					
fairness index	0.986	0.779	0.876	0.897	0.978
router loss rate	0.0034	0.0180	0.0283	0.0486	0.0825
perceived loss rate	0.0029	0.0117	0.0135	0.0267	0.0504
burst loss rate	0.000150	0.001458	0.001894	0.005198	0.018356
bandwidth exploitation	0.579	0.773	0.766	0.857	0.942

Figure 3: Various Protocol Instances sharing the same bottleneck, Fairness Index, Loss Rate at the router, Perceived Loss Rate, burst Loss Rate and Bandwidth Exploitation. Average values on all the instances. Top: RED router, bottom: Drop-Tail router.

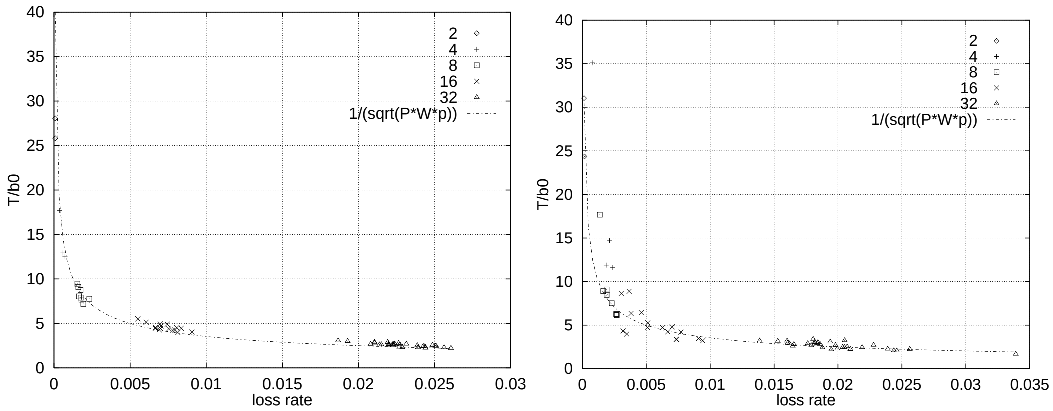

Figure 4: Throughput versus loss-rate, RED (left) and Drop-Tail router (right).

rate) is much smaller than the previous ones. This is because the loss rate has a bursty distribution, that is a side effect of the leave delay.

The last parameter shown in table 3 is the average bandwidth utilisation of the aggregate flow. As anticipated in section 3, the coarse quantisation of allowed throughputs sometimes leads to a non optimal utilisation of the available bandwidth. This is more evident with fewer protocol instances which stabilise on high throughput levels, characterised by larger rate steps between one level and the adjacent ones.

Figure 4 shows the dependency of the throughput on the burst loss rate. Also the approximation of formula 2 is drawn. Experimental data confirm the model validity, showing the same dependency on the loss rate. The data points obtained with RED routers, fit very well in the model prediction and show a tight grouping of points belonging to the same run, confirming a good behaviour of the protocol.

The second experiment has been run to see the degree of fairness of our protocol with the respect to TCP connections. Referring to topology (a) in figure 2, we have run 8 instances of our protocol and 8 TCP connections on the two paths n_0—n_5 and n_4—n_3. Our protocols instances have been evenly distributed across the two paths while all the TCP

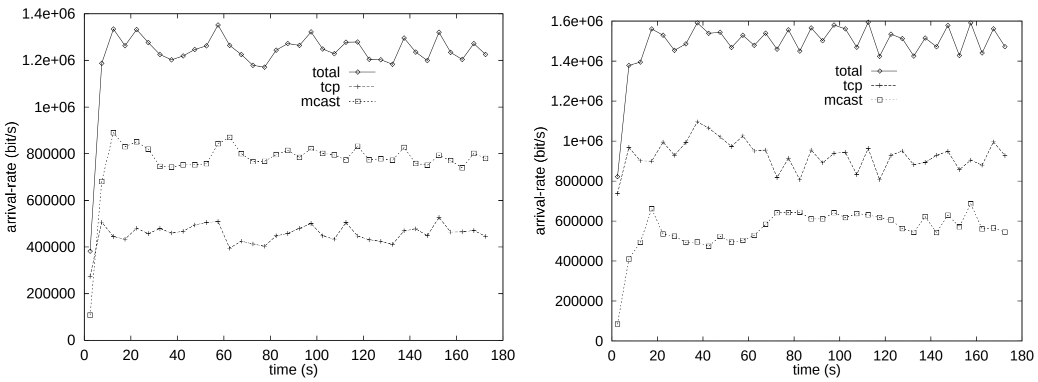

Figure 5: Bandwidth sharing with TCP, RED router, TCP MTU size 256 bytes (left) and 1024 bytes (right).

connection have been placed on the path n_0—n_5, characterised by 420ms RTT. All the starting time have been determined randomly in the interval [0, 10]s. Figure 5 shows the aggregate throughput of TCP connections and that of our protocol instances. The graph on the left refers to TCP with 256 bytes packet size, while the one on the right refers to 1024 bytes packet size. According to equation 3 TCP with 256 bytes packet size and 420ms RTT should be as aggressive as our protocol (with the usual parameter setting). However figure 5 shows that our multicast protocol is slightly more aggressive, that is due to the fact that our algorithm tends to aggregate burst losses more than TCP does, as already anticipated in 3.3.1. Anyway an average TCP connection, even with large RTT and the usual (large) packet size, is more aggressive than our protocol, with the current parameters setting.

The last experiment we have run concerns the synchronisation of receivers belonging to the same protocol instance. In pursuit of evaluating this feature, we run a simulation on the topology (b) in figure 2. We placed a sender on node n_0 and three receivers (r_0, r_1 and r_2) respectively on nodes n_3, n_4 and n_5, starting them about 6 seconds apart. Background traffic has also been set on the bottleneck n_1—n_2 starting other 5 protocol instances on the paths n_0—$n3$ and n_0—n_4. Figure 6 shows the subscription level of the three receivers, and the losses they experience. Receiver r_0 start first and quickly[10] reach L_4, the same is done by r_1 about 6 seconds later. After about 1 second r_1 has reached L_4, they both experience losses and fall back to L_3 synchronising. A similar history happens to r_2 which reach the synchronisation point (again a loss event) at time 26.

5 Conclusions and Future Works

A protocol for one to many bulk-data transfer on the MBone has been presented. It is reliable with the respect to packet loss and embodies a receiver-driven congestion control mechanism, based on a cumulative layered organisation of the data packets being transmitted. A redundant encoding scheme has been used both for reliability purpose and for allowing an effective layered organisation. The protocol is characterised by a full scalability with the respect to the number of receivers, accomplished by moving the tasks of congestion control and packet loss recovery in the receivers. The protocol has been implemented in a prototype and its performance evaluation has been done by means of simulations, which

[10] Note a feature of the protocol not discussed in the specification: provided that the bottleneck bandwidth can be evaluated, at the startup receivers increase exponentially their throughput (every P bursts instead of $P * 2^i$) up to a given fraction of the bottleneck bandwidth. Bandwidth evaluation is made on the first burst using the packets transmitted back-to-back.

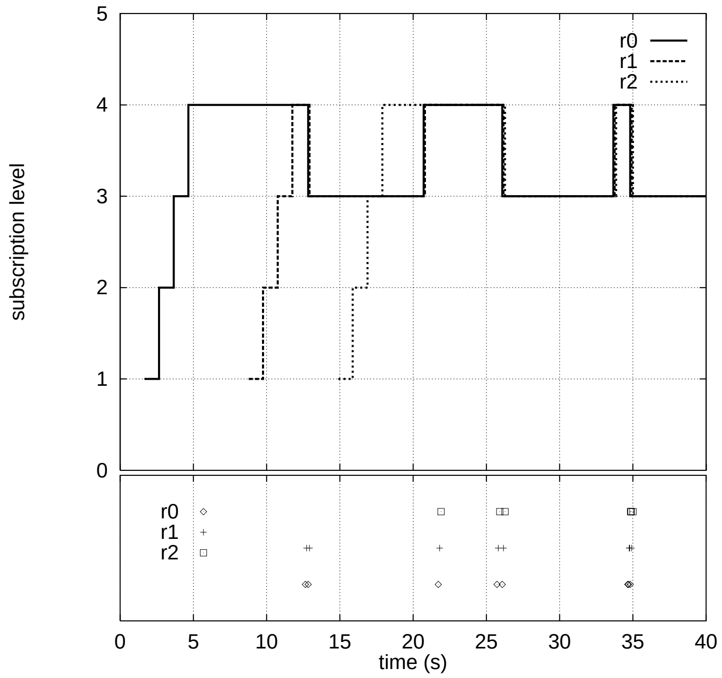

Figure 6: Synchronisation of three receivers belonging to the same protocol instance.

have shown the protocol to present fairness among its own instances and tunable fairness with TCP.

Future works on this subject will include a more extensive simulated evaluation of protocol performances using larger network topologies and more complex scenarios. We have also planned to test the protocol directly in ad-hoc testbeds and in the real MBone. We are finally interested in evaluating the usefulness of the protocol as a transport layer of existing applications, such as Web mirroring.

The authors whish to thank Luigi Rizzo for the useful discussions, concerning congestion control issues, and for providing with an implementation of the redundant encoder/decoder.

References

[1] D-M.Chiu, R.Jain "Analysis of the Increase and Decrease Algorithm hms for Congestion Avoidance in Computer Networks", *computer Networks and ISDN Systems*, V.17, pp.1-14, 1989.

[2] K.Fall, S.Floyd, "Simulation Based Comparisons of Thaoe, Reno, and SACK TCP", CCR, Vol.26 N.3, Jul. 1996, pp.5-21.

[3] W. Fenner, "Internet Group Management Protocol, Version 2", INTERNET-DRAFT (working draft), Jan. 20, 1997, `http://ds.internic.net/internet-drafts/-draft-ietf-idmr-igmp-v2-06.txt`

[4] S.Floyd, K.Fall, "Router Mechanisms to Support End-to-End Congestion Control", Technical report, `ftp://ftp.ee.lbl.gov/papers/collapse.ps`.

[5] J.Hoe, "Stat-up Dynamics of TCP's Congestion Control and Avoidance Schemes", Jun. 1995, Master's Thesis, MIT.

[6] V.Jacobson, "Congestion Avoidance and Control", *ACM SIGCOMM'88*, August 1988, Stanford, CA, pp.314-329.

[7] V.Jacobson, "Modified TCP Congestion Avoidance Algorithm", Tech. report, 30 Apr. 1990. Email to end2end-interest Mailing List, `ftp://ftp.ee.lbl.gov/email/-vanj.90apr30.txt`.

[8] S.McCanne, V.Jacobson, M.Vetterli, "Receiver-driven Layered Multicast", *ACM SIG-COMM'96*, August 1996, Stanford, CA, pp.1-14.

[9] UCB/LBNL Network Simulator - ns - version 2, 1997, `http://www-mash.cs.berkeley.edu/ns/`.

[10] B. Kantor and P. Lapsley, "Network news transfer protocol: A proposed standard for the stream-based transmission of news," Request for Comments (Proposed Standard) RFC 977, Internet Engineering Task Force, Feb. 1986.

[11] T.Ott, J.Kemperman, M.Mhathis, "The stationary distribution of ideal TCP congestion avoidance", Technical Report, Aug. 1996.

[12] L. Rizzo, "Effective erasure codes for reliable computer communication protocols", CCR, V.27 N.2, April 1997, pp.24-36.

[13] L.Rizzo, L.Vicisano, "A Reliable Multicast data Distribution Protocol based on software FEC techniques", *The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97)* Sani Beach, Chalkidiki, Greece June 23-25, 1997 (to appear).

[14] H.Schulzrinne, S.Casner, R.Frederick, V.Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet Official Protocol Standards, RFC 1889, Jan 1996, `http://ds.internic.net/rfc/rfc1889.txt`.

[15] W. Stevens., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.", RFC2001, January 1997.

[16] L.Vicisano, "Notes on a cumulative layered organisation of data packets across multiple streams with variable-rate", `http://www.cs.ucl.ac.uk/staff/L.Vicisano/-layers.ps`.

[17] L.Vicisano, M.Handley, J.Crowcroft, "B-MART, Bulk-data (non-real-time) Multiparty Adaptive Reliable Transfer Protocol", `ftp://cs.ucl.ac.uk/darpa/bulk.ps.Z`.

[18] D.Wessels, K.Claffy, "Internet Cache Protocol (ICP), version 2", INTERNET-DRAFT (working draft), May 27 1997, `http://ds.internic.net/internet-drafts/-draft-wessels-icp-v2-03.txt`.

[19] D.Wetherall and C. Lindblad, "Extending Tcl for Dynamic Object-Oriented Programming", *Proceedings of the Tcl/Tk Workshop '95*, Toronto, July 1995.

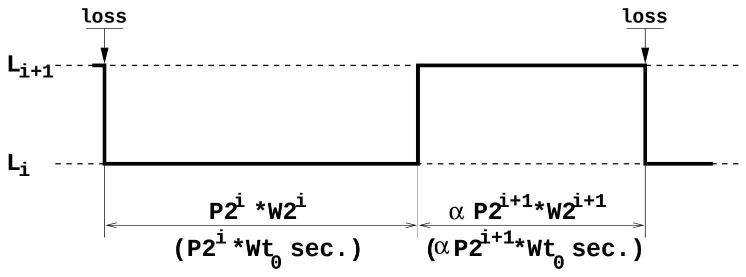

Figure 7: Steady state model.

A Simplified Model of the Receiver

To compute the dependency of the throughput on the loss rate, we use a simplified model of our congestion control algorithm, assuming that receivers reach a steady state, where they oscillate between two subscription levels: i and $i+1$. Experimental observations have shown this to be a realistic assumption, and we will see that the results we obtain from this model conform to the simulation outcome.

Referring to figure 7, we consider a receiver that, after experiencing a loss, decreases its subscription level from L_{i+1} to L_i, leaving layer C_{i+1}. Then, after having seen $P * 2^i$ bursts in L_i, it subscribes again L_{i+1}, remaining in this state for $\alpha * P * 2^{i+1}$ until experiences another loss. Note that it must be $0 \leq \alpha \leq 1$, otherwise the receiver would subscribe L_{i+2} violating the model assumptions.

The throughput associated to such a steady state is the average throughput of L_i and L_{i+1} weighted to the time the receivers spends in each level:

$$T = b_0 * 2^i \frac{1+4\alpha}{1+2\alpha} \tag{4}$$

On the other hand, the number of packets received in a complete cycle $(W2^i * P2^i + \alpha W2^{i+1} * P2^{i+1}$ must be, on average, equal to the the number of packets between two consecutive losses $(1/p)$. Imposing this we obtain:

$$2^i = \frac{1}{\sqrt{p} * \sqrt{P * W(1+4\alpha)}} \tag{5}$$

Substituting (5) in (4) we obtain:

$$T = \frac{\sqrt{1+4\alpha}}{1+2\alpha} \frac{b_0}{\sqrt{p}\sqrt{P * W}} \tag{6}$$

Although expression 6 is a function of α, we can determine a narrow range of allowed values for T, being the term $\frac{\sqrt{1+4\alpha}}{1+2\alpha}$ bounded between $\sqrt{5}/3$ and 1 (with $0 \leq \alpha \leq 1$). Thus we have:

$$T_{max} = \frac{b_0}{\sqrt{p}\sqrt{P * W}} \tag{7}$$

$$T_{min} = \frac{\sqrt{5}}{3} T_{max} \tag{8}$$

So far we have not taken into account the leave delay, it has the effect of increasing the actual throughput; in fact, after the layer C_{i+1} has been left, packets transmitted in it still keep being received for the leave delay length. To validate our model, we have computed the throughput discarding the packets received from left layers. In figure 8, that shows

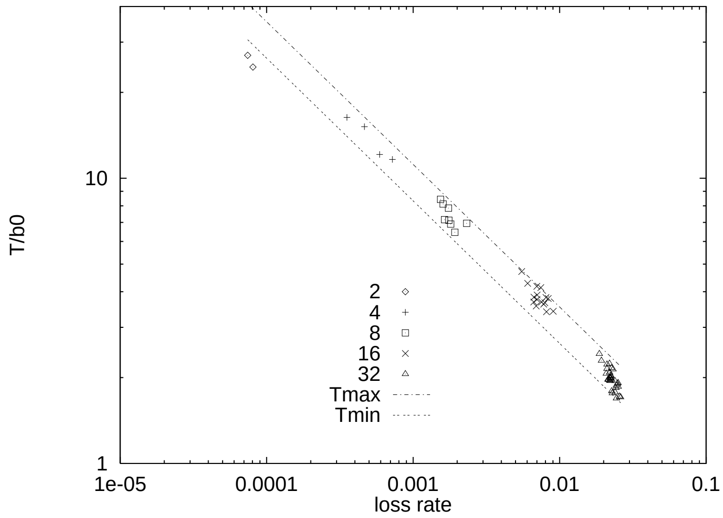

Figure 8: Comparison between simulated results and model prediction.

the dependency of throughput, computed as above, on the burst loss rate, we can see that the simulated results fit well our model. Points concerning the run with 2 protocol items probably have a too small loss rate to be statistically meaningful.