Appeared in Proceedings of IEEE INFOCOM'98, the Conference on Computer Communication, San Francisco, USA, March 29–April, 1998. Also UCL-CS Research Note RN/97/67.

TCP-like Congestion Control for Layered Multicast Data Transfer

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

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

Abstract—**We present a novel congestion control algorithm suitable for use with cumulative, layered data streams in the MBone. Our algorithm behaves similarly to TCP congestion control algorithms, and shares bandwidth fairly with other instances of the protocol and with TCP flows. It is entirely receiver driven and requires no per-receiver status at the sender, in order to scale to large numbers of receivers. It relies on standard functionalities of multicast routers, and is suitable for continuous stream and reliable bulk data transfer.**

In the paper we illustrate the algorithm, characterize its response to losses both analytically and by simulations, and analyse its behaviour using simulations and experiments in real networks. We also show how error recovery can be dealt with independently from congestion control by using FEC techniques, so as to provide reliable bulk data transfer.

Keywords: congestion control, multicast.

I. Introduction

The design of one-to-many data transfer protocols which run on top of the Internet multicast service have to face the fundamental problem of *congestion control*. The network is shared by many competing users, and uncontrolled, aggressive flows can bring the network to a congestion collapse. To avoid this, protocol instances should behave in such a way to achieve a fair sharing of network resources with instances of other well-behaved protocols. At the same time, in multicast protocols, effective congestion control should not come at the expense of scalability, since we wish to address scenarios comprising thousands of receivers.

The problem is becoming more and more important with the increasing diffusion of bandwidth-intensive multimedia tools for video and audioconferencing, which stimulate the use of the network for reaching large, heterogeneous groups of users. For these applications, techniques have been proposed [15], [16] for transmitting the same stream using multiple data rates (which map into different quality levels) based on a cumulative layered data organization. By using proper data arrangements [17], a similar approach can be used for the reliable transfer of bulk data.

Multicast routing provides the basic mechanism for distributing data with different rates to subsets of the multicast distribution trees. By joining/leaving multicast groups, receivers in a subtree can in principle adapt the data rate to the available network capacity. But this approach can only work if receivers are able to evaluate and adapt to the network capacity, act in coordination, and if the algorithms that are used can scale to large groups of receivers. The requirement of being a fair competitor with other well-behaved protocols such as TCP further complicates the problem of devising a suitable algorithm for letting receivers dynamically choose the appropriate data rate.

In this paper we will describe a receiver driven congestion control algorithm which is TCP-friendly and suitable for use in the Mbone. We support different bandwidths by using a layered organization of data, and letting receivers adapt to the available bandwidth by joining to one or more multicast groups. Each receiver takes decisions autonomously, but techniques are used to synchronise receivers behind the same bottleneck (and belonging to the same protocol instance), and achieve an effective use of the bandwidth. No explicit forms of group membership are used, making the algorithm fully scalable and simple to implement, both at the sender and the receiver side.

Previous works on this subject, such as [9], have already used layered data organization to implement some form of multicast congestion control. The novel results of our work are the development of a congestion control algorithm which competes fairly with TCP flows, and the achievement of receiver synchronisation without recurring to the use of explicit group membership protocols. Other significant contributions of the paper lie in the evaluation of the algorithm and its interaction with TCP using analytical techniques, simulations, and experiments in real networks.

The paper is structured as follows. In Section II we briefly describe the basic mechanisms our congestion control algorithm is based upon, namely the relation between throughput and loss rate, multicast group membership in the Mbone, and layered data organization. Section III describes in detail our congestion control algorithm, while in Section IV we prove its effectiveness using analytical techniques, simulations and experiments in real networks. Finally, Section V briefly discusses the application of the algorithm to continuous streams, and then shows techniques to achieve reliable bulk data transfer while preserving scalability and optimizing the usage of the network.

II. Basic mechanisms

In this Section we briefly review the three basic mechanism our congestion control algorithm is based upon.

A. Relation between throughput and loss rate

IP networks do not have an explicit congestion notification mechanism. The response to congestion is dropping packets, and the loss of one or more packets at the receiver is reported to the sender where it is interpreted as a congestion signal. A congestion control algorithm must react to congestion signals by reducing the transmission rate. The relation between the packet loss rate, p, and the throughput, T, of a well behaved session is such that increasing values of p correspond to decreasing values of T. For the congestion control algorithms used in TCP [6], [14], various authors have derived the following approximate relation between throughput and loss rate:

$$T = \frac{s \cdot c}{RTT\sqrt{p}} \tag{1}$$

where s is the packet size, RTT is the round trip time for the session, and the constant c is in the range 0.9...1.5 [5], [8], [11]. The relation holds for small values of p; at high loss rates, TCP tends to lose its ACK-based self clock, the session assumes a chaotic behaviour, and the actual throughput has a sharp decrease and becomes highly dependent on the length of the retransmission timeouts.

Equation 1 shows that, in presence of competing receivers behind the same bottleneck (thus experiencing a similar loss rate), the actual share of the bandwidth for each of them strongly depends on s and RTT. Apart from this variance, competing TCP sessions all reduce their throughput similarly in response to losses. Fairness among different protocols requires that all of them have a similar behaviour in response to losses, with comparable functional dependencies, absolute throughput values, and response times.

Equation 1 does not capture completely the behaviour of the control system. Another important parameter of any control algorithm is the speed of response to variations in the controlled system – in our case, in network conditions. For TCP congestion control, this response time is in the order of the RTT of the session, which is the time required for the sender to receive the feedback and react.

B. Multicast group membership

Forwarding of multicast data in the Mbone is implemented by multicast routers, which communicate with hosts using a dedicated group membership protocol, IGMP [3], [4]. Packets for a given multicast group are forwarded into a network segment only if that segment leads to active receivers for that group (be them end nodes or other multicast routers serving active receivers).

In the *join* phase, a receiver joining a new group sends an IGMP message to the router, which will in turn enable forwarding for that group, possibly propagating the information to the upstream router (*graft* message). The join phase is fast, taking roughly one RTT (computed between the joining node and the first router towards the source which is blocking the group).

In the *leave* phase, a receiver informs its local router that it is not interested in the group anymore. This triggers a polling phase in which the multicast router checks if other receivers in the local subnet are still interested in the group. If no active receivers exist, the router sends a *prune* message for the group to the upstream multicast router, so that forwarding for that group is blocked. Since the absence of active receivers can only be determined after a timeout, the leave phase can take a considerable time (a few seconds) which we call the *leave delay*.

By joining and leaving a group, a set of receivers acting in coordination can exercise a simple on/off control the flow data in a subtree, although the minimum duration of the on period is constrained by the leave delay.

C. Layered organisation of data

Our congestion control protocol is aimed to the distribution of the same data (possibly, with different quality) to a set of receivers with different, increasing bandwidths B_i's, $i = 0 \ldots l-1$. The use of l different multicast groups, each with bandwidth B_i, would make a very poor use of the network capacity. The problem can be solved in a more efficient way by using a cumulative, layered data organisation (*layered data organisation* for brevity) based on l multicast groups or *layers*. Each layer carries data with bandwidth L_i, in such a way that $B_i = \bigcup_{j=0}^{j=i} L_j$. Each receiver can tune its receive bandwidth by joining the appropriate number of layers, and there is no waste of bandwidth if receivers with different requirements share the same links.

A layered data organisation is only possible if the data to be transferred supports it, i.e. $\forall i$, the data carried with bandwidth B_i is a subset of the data carried with bandwidth B_{i+1}. This is usually feasible for multimedia data streams; for bulk data such as a file, suitable data arrangements which minimize replications of data are shown in [17]. In both cases, however, the problem of finding a suitable data organisation becomes more and more complex as the number of layers grows. This, and scalability reasons, suggest that only a small number of values for B_i are made available from the transmitter, e.g. exponentially spaced; receivers can then choose the *subscription level* which best matches the available bandwidth.

III. Congestion control for multicast layered data

Our congestion control mechanism is designed for a transmitter sending data to many receivers in the Mbone. The communication might involve a continuous stream (e.g. video or audio data), or a reliable, bulk data transfer (e.g. a file). In this Section we will ignore the fact that lost packets might need to be retransmitted in order to complete a data transfer; this problem will be tackled in Section V.

In unicast communications, the sender takes part to congestion control by changing its sending rate according to the congestion signals that it receives. In multicast communications, this approach would be problematic, since groups of receivers with different requirements may exist, and adapting to the needs of one set of receivers would penalize others with contrasting requirements.

As in other proposals appeared in the literature [9], we achieve the desired effect by using the multicast router driving the bottleneck to modulate the sending rate for the subtree. This is possible thanks to the layered data organisation, which gives receivers the possibility to modulate the receive rate by joining/leaving layers. Adaptation to heterogeneous requirements becomes possible (and simple) because it can be done independently on each subtree served by a multicast router. As a consequence of this approach, the transmitter does not even need to take part in congestion control (except for using a layered data organisation).

In our algorithm, receivers try to infer the status of the network using congestion signals (typically, packet losses), and control the flow of incoming data using the pruning capability of IP multicast routing. Each receiver joins or leaves layers depending on the received congestion signals. The strategy to join or leave layers is chosen in such a way to emulate the behaviour of TCP and generate a similar relation between throughput and loss rate. Intuitively, a receiver with subscription level i will behave as follows:

> if a loss is experienced, decrease the subscription level;
> otherwise, if no losses are experienced for a time $t_p(i)$,
> increase the subscription level.

With a proper choice of the bandwidth B_i corresponding to each subscription level, and the delay $t_p(i)$ before moving to the next level, the above algorithm mimics the behaviour of TCP (additive increase, multiplicative decrease [2]) and yields a relation between throughput and loss rate similar to that of TCP, namely $T \propto 1/\sqrt{p}$. Parameters can also be tuned to yield a response time comparable to that of TCP congestion control.

Unfortunately, we also have to deal with some additional problems, not present in unicast communications, for which the simple algorithm presented above is not adequate. First, congestion control cannot be effective if receivers behind the same router act in uncoordinated way. Second, the length of the leave delay makes *failed* joint attempts (i.e. joins to a layer which cause the bandwidth to exceed the bottleneck bandwidth) have long lasting effects. Both phenomena can severely compromise the effectiveness of the algorithm, so we need to develop appropriate countermeasures.

A. Synchronisation points

With more than one receiver behind a bottleneck, synchronization among receivers is fundamental in order to make the pruning mechanism work. In fact:

(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 might interpret the resulting 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;

All these problems can be minimised if join attempts from different receivers are coordinated. Unfortunately, receivers tend to diverge because of different start times, propagation delays, loss patterns, and the history of events experienced by each of them. We have avoided the introduction of an explicit protocol for receiver coordination, since it could pose scalability problems, and could be slow to converge. Rather, we have used an implicit coordination protocol based on the use of *synchronisation points* (SP's).

SP's correspond to specially flagged packets in the data stream. A receiver can only make a join attempt immediately after an SP, and can only base its decisions on the history of events since the last SP. Both strategies help in keeping receivers in synch, especially for nearby nodes which will be less affected by skews in propagation delays and losses caused by background traffic. The deletion of history across SP's serves to avoid that random events experienced by each receiver accumulate their effects and cause receivers to diverge.

The distance between SP's on each layer is related to the time $t_p(i)$ that a receiver must spend at a given level before doing a join attempt. The phenomenon illustrated in point *(b)* above suggests that when a receiver at level i does a join attempt, all receivers at the same or lower levels might experience the losses related to a failure. To overcome this problem, SP's at each level are always a subset of the SP's at the previous level. This has two implications. First, when a receiver does a join attempt, all receivers at the same or lower levels will do an attempt as well; as a consequence, any congestion signal will not cause further divergent behaviours. Second, receiver with a lower subscription level have more chances to increase the level and aggregate to other receivers at a higher level.

B. Sender-initiated probes

The problems described in the previous section are further aggravated by the length of the leave delay, which makes a join attempt (and especially, a failed one) have consequences on the network for a long time. Join attempts (and consequently, SP's) should then be sufficiently far apart to let the network settle in between. Furthermore, as an optimisation of the algorithm, we would like to be able to increase the subscription level only when there is confidence that the attempt can be successful.

Our algorithm involves a form of sender-initiated probe for bandwidth, consisting in the periodic generation of short bursts of packets, followed by an equally long relaxation period during which no packets are sent. The bursts have the effect of a join attempt. For the duration of the burst, the bandwidth is effectively doubled; if the bottleneck bandwidth is not sufficient, queues build up and possibly cause packet losses, or an increase in the packet interarrival times. Such congestion signals are not interpreted as a signal to lower the subscription level, but rather as hints for *not* increasing the subscription level. The advantage of these sender-initiated probes over real join attempts is that the surge of packets that would follow the failed join attempt will not last for the whole leave delay, but just for the (much shorter) duration of the burst, and recovery will be possible during the subsequent relaxation period. We note that sender initiated probes also mimic the behaviour of TCP sessions, which, in a number of occasions (e.g. when opening windows, or receiving an ACK for more than one MSS) transmit two or more back to back packets.

C. The deaf period

The leave phase takes a long time to complete, even when all receivers in a subtree drop the layer which caused congestion. A naive application of the algorithm presented in Section III would probably result in decreasing the subscription level multiple times in response to a failed join attempt. To overcome this problem, we have introduced a *deaf period* t_D . After a loss, and the subsequent decrease in the subscription level, a receiver does not react to further losses for a time t_D, in order to account for the slow response of the multicast router to leave requests. The deaf period is set slightly larger than the leave delay, so as to give the multicast router sufficient time to respond to the leave request.

D. The congestion control algorithm

In this Section we provide the full description of our congestion control algorithm, with the addition of the mechanisms discussed so far. We assume that l different bandwidths are offered, $B_i, i = 0 \ldots l-1$, with increasing values (e.g. exponentially spaced by a factor of 2). Data is transmitted over l multicast groups with bandwidths L_i, where $L_0 = B_0$ and $L_i = B_i - B_{i-1}, i = 1 \ldots l-1$. For each bandwidth B_i packets are normally transmitted at a constant[1] rate, appropriately placed onto the different layers; we call τ_i the inter-packet time at each layer i. Two additional parameters, W and P, control the behaviour of the algorithm: W represents the distance between bursts, in multiples of τ_0, P represents the number of bursts between synchronization points in layer L_0.

During bursts, which have a duration τ_0, two back-to-back packets are sent at each transmission time. Following the burst, there is an interval τ_0 during which the transmission is suspended. Finally, an interval $(W-2)\tau_0$ follows during which transmissions occur at the nominal rate.

Synchronisation points are spaced proportionally to the bandwidth corresponding to the subscription level, and are located at the end of a burst (which corresponds to the last packet of the burst in layer L_0). As a reference, Figure 1 shows the sequence of packet transmissions for $P = 1, W = 8$ and five layers, and the location of synchronisation points.

The distance between SP's at subscription level i is $(B_i/B_0)PW\tau_0$. The constants τ_0, P and W appear in the relation between throughput and loss rate, and will be discussed in Section IV-B.

Subscription levels can increase only at SP's, and decrease at any time, using the following rules:

- *decrease* if a loss is experienced during normal transmissions (except for a time t_D after a previous decrease);
- *increase* at a SP if no losses are experienced during the burst preceding that SP;
- *unchanged* otherwise (this includes the case of losses experienced during a burst, or during the deaf period).

The deaf period t_D after a decrease serves to avoid cascaded losses while the leave can effectively complete.

It should be noted that the protocol does not depend heavily on the bandwidths being exponentially distributed, or on the use of probes (although probes do show a performance improvement in our simulations). The main, and novel, elements of the algorithm are the use of synchronisation points and the rules for changing subscription levels.

IV. Performance analysis

In this Section we evaluate the performance of our congestion control algorithm by means of analytical techniques, simulations using the `ns` simulator [10], and experiments on real networks using a prototype implementation of the algorithm. We have investigated the start up and steady state behaviour, adaptation to changing network conditions, fairness among protocol instances and with TCP instances. The network topologies used in the simulations are shown in Figure 2, with transmitters and receivers spread on the various nodes. Experiments (both simulation and real measurements) generally include multiple instances of our multicast congestion control algorithms, competing TCP protocol instances, and background traffic. The protocol we evaluated used exponentially distributed bandwidths and parameters $P = 1, W = 8$. Due to space limitations, we only present the main results of our analysis; more detailed information is present in [18].

[1] depending on the type of data, this might only apply to the average rate

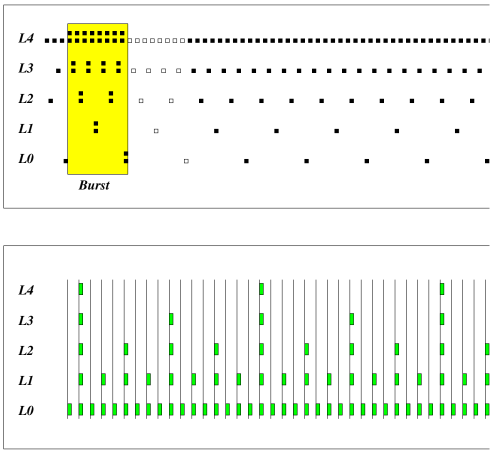

Fig. 1. The transmission pattern at different layers, using $P = 1, W = 8$ and five layers. Above: the sequence of packet transmissions between two bursts, indicating the arrangement of packets on the various layers, the burst and the subsequent recovery time (white squares). Below: the location of synchronisation points at the various layers; the vertical lines represent the location of bursts.

A. Throughput versus loss rate

The relation between throughput and loss rate can be derived using an approximate steady state model of the system, in which a receiver oscillates among two subscription levels, i and $i+1$, which surround the "correct" bandwidth. In the approximate model, the throughput depends on the fraction α of time which is spent at level $i+1$, which in turn depends on the loss rate experienced by the connection. In [18] we derive the following approximation for the throughput:

$$T \approx \frac{s}{\tau_0} \frac{c'}{\sqrt{p}\sqrt{PW}} \tag{2}$$

where s is the packet size, τ_0 is the inter-packet time at layer L_0, W is the number of packets between two bursts, P is the distance (in bursts) between synchronisation points at the lower level, and $c' \in [1,2]$ is a function of α. p represents the burst loss rate, computed accounting closely spaced (i.e. closer than t_D) losses as a single one. Note that $s/\tau_0 = B_0$, but we have left s and τ_0 in the equation to remark its similarity to the same relation for TCP. Apart for a multiplying constant, Equations 1 and 2 reflect the same behaviour in response to losses.

B. Parameter tuning

Parameters W, P, and B_0 influence the actual response to losses of our algorithm (both steady state and dynamic), and they should be tuned to achieve the same response to losses as TCP. TCP sessions have a wide range of RTT's, with common values for non-local communications in the 0.1-1 s range. For the reasons we will discuss in the following paragraphs, it is difficult for us to achieve very short response times, so we can only aim for the high end of the range. i.e. a response time of about 1 s.

W is the distance between bursts, measured in packets transmitted at the lowest level. W cannot be less than 2, but in practice it needs to be larger in order to make bursts sufficiently far apart. On the other hand, W should not be too large or it would make synchronisation points too distant, hence making the algorithm slower to converge to the correct operating point. In our experiment we have used $W = 8$

P represents the distance between synchronisation points at the lowest level, measured in bursts (the same distance, measured in packets, is PW). The contrasting requirements of making W large and synchronisation points not too distant almost unavoidably forces us to use $P = 1$, unless we work with very high bandwidths.

$B_0 = s/\tau_0$ is the lowest bandwidth that we can use on our session. There are some practical lower bounds on its acceptable values. The first one comes from the application, in that some data streams (e.g. audio or video) cannot be transferred with an acceptable quality below a certain bandwidth. A second lower bound is related to the speed of response of the congestion control protocol. Since the distance between synchronisation points is at least PW packets, lowering the bandwidth also means increasing the distance between synchronisation points. Reducing the packet size is only possible to a certain extent, above which the overhead for packet headers becomes unacceptably large.

In our experiments, we have used a packet size $s = 256$ bytes. Our target of 1 second between synchronisation points at the lowest subscription level, and the minimum values for P and W of 1 and 8 respectively, translate into a minimum bandwidth of 2 KB/s.

C. *Differences with TCP*

Although our algorithm tries to mimic the behaviour of TCP congestion control, and Equations 1 and 2 only differ by a multiplying constant, there are some differences between the behaviours of the two algorithms.

First of all, with our algorithm, the relative throughputs for competing sessions experiencing the same loss rate does not depend on the “distance” (RTT) between the sender and the receiver. In this respect, TCP congestion control has what can appear as a degree of unfairness, in that the throughput is inversely proportional to the RTT. In fact, both approaches are appropriate having different targets:

- In TCP, congestion control is completely controlled by the sender, and the only way the receiver has to reduce throughput is to delay the generation of acknowledgements, making the RTT appear larger than it really is. Furthermore, connections with a larger RTT are less responsive than those with a shorter RTT. For both reasons, it is more than acceptable that bandwidth is not allocated evenly to sessions with a different RTT, and this should not be seen as a sign of unfairness.
- Our congestion control algorithm lets receivers control throughput via the join/leave mechanism, so there is no need for an additional mechanism such as the dependency on the RTT. Besides, the response time of the algorithm does not depend on the RTT but rather on (usually longer) times such as τ_0 and, especially, the leave delay, so it is reasonable that all protocol instances get the same share of bandwidth irrespective of the RTT.

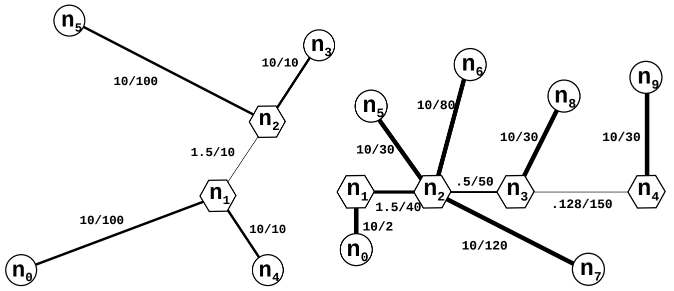

Fig. 2. The network topologies used in the simulations.

A second difference is that our protocol works by controlling the data rate, while TCP congestion control is based on the ‘conservation of packets’ principle [6]: a new packet is not put into the network until an old packet leaves; this allows TCP to fine-tune its traffic generation and to react quickly in the face of short-term network load changes. On the other hand it heavily relies on the ack-based self-clock mechanism to work, making TCP break at high loss rates, as discussed below.

Another important difference is that our algorithm has a limited set of allowed bandwidths (exponentially distributed), while TCP is able to adjust its bandwidth which much finer granularity. This has two major consequences: one is that we might obtain a worse exploitation of the available bandwidth (but this is not a big issue as we will discuss in the following); the other is that the minimum allowed bandwidth can be higher than what the network can sustain, in which case we can have high loss rates.

TCP also has a minimum bandwidth (roughly one packet per RTT) below which the throughput has a sharp decrease and connections assume a chaotic behaviour, being affected by large retransmission timeouts. It could be possible, in principle, to implement a behaviour similar to TCP even in our algorithm, so that when excessive losses are experienced at subscription level 0, a receiver disconnects even from layer 0 to resume reception at a later time. Of course, all receivers behind the same bottleneck should act synchronously, for the reasons explained in Section III-A.

However, the time constants involved in this mechanism must be extremely long (several seconds at the very best). This can make the resulting quality unacceptable for some applications (e.g. audio or video) suggesting a permanent, rather than temporary, disconnection. Furthermore, experience with the use of Web browser on congested networks suggests that the user will always try to override timeouts when they become too long (e.g. pushing the 'Reload' button, or opening new connections), thus defeating the mechanism. As a consequence, we believe that the proper approach to deal with high losses lies in the use of appropriately low values of B_0, and in the use of queue management mechanisms in the routers to deal with uncontrollable flows [5].

D. *Validation of the model – inter-protocol fairness*

To validate our analytical model, we have both simulated the behaviour of several instances of the protocol sharing a bottleneck link, and run an implementation of our protocol on a real

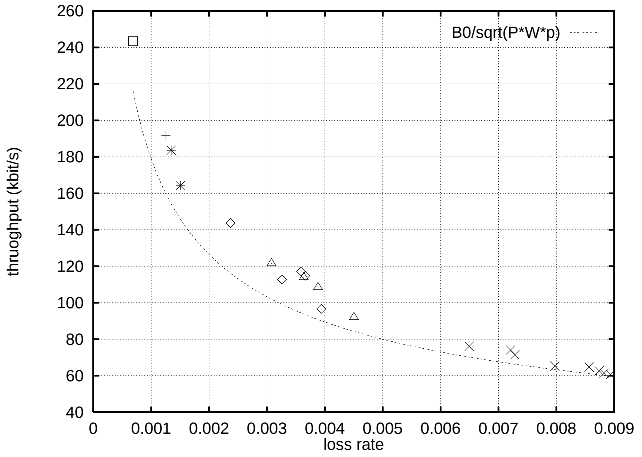

Fig. 3. The relation between throughput and (burst) loss rate in real measurements, compared to the analytical results from Eq. 2 (solid line). Protocol instances run concurrently are represented by the same symbol.

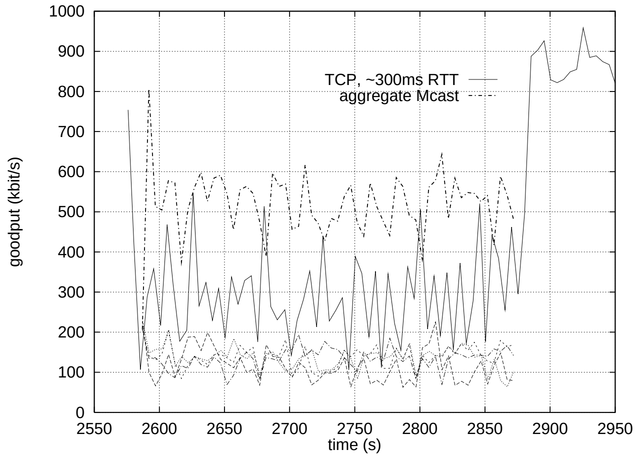

Fig. 4. Bandwidth sharing among one TCP connection (top) and 4 protocol instances (bottom) on a real network. $RTT \simeq 300$ms, and TCP packet size of 1440 bytes.

Internet path. For these simulations we have used the topology of Figure 2,left, with both RED and tail-drop routers. We spread N pairs of sender/receiver on the two paths with different delays (n_0—n_5 and n_4—n_3). Senders were started at random times over an interval of $N/2$ seconds, and receivers had been kept active for 300 seconds. The results (see [18] for more details) matched very closely the model. In particular, simulations using RED routers produced values very close to the curve representing Eq. 2, whereas simulations with tail-drop routers gave slightly higher throughputs, similar to Fig. 3 (which refers to experiments on a real network).

A similar experiment has been done using an implementation of the algorithm on a real Internet path connecting two subnets, one in UCL (London) and the other in DEIT (Pisa). The path is characterized by a maximum throughput slightly greater than 1 Mb/s, average RTT between 70 ms (unloaded) and 300 ms (loaded), traversing 15 nodes in both directions (the route has been stable for the whole duration of the experiment). Since traffic on the Mbone is generally not congestion controlled, in order to get significant results (i.e. compete with other congestion-controlled traffic) we had to set a multicast tunnel, without throughput limitations, between two multicast routers feeding the two subnets. We have run from one to eight concurrent instances of the protocol, sharing the links with the normal Internet traffic. Results are shown in Figure 3, and are in accord with the model prediction and with simulations. The actual throughput, slightly higher than the analytical value, can be motivated by the (probable) use of tail-drop routers in the network, which cause more bursty losses.

The experiments presented above have also been used to evaluate the fairness in sharing bottleneck bandwidth among several instances of the same protocol. From Figure 3 we see that protocol instances running concurrently present a similar throughput. To evaluate better this feature, we have computed the fairness index[2] [2] from the results of the simulations. Simulations with 2 to 32 protocol instances resulted in fi in the range 0.978..0.998 with RED, and 0.779..0.986 with tail drop. In both cases, the fi values are satisfactory, and particularly good with RED.

Another parameter determined in simulations was the fraction of available bandwidth used by the aggregate flow, ranging from 0.57 with 2 instances to 0.94 with 32 instances. The coarse quantisation of allowed throughputs sometimes leads to a non optimal utilisation of the available bandwidth, and the phenomenon is obviously more evident with fewer protocol instances. However this is not a significant problem on a typical backbone link, where multiple flows typically exist which can use the residual capacity.

E. Fairness with TCP

A second set of simulations and experiments has been run to investigate the behaviour of our protocol when competing with other TCP instances. Using the ns simulator, we have run 8 multicast protocol instances as in the previous experiments, together with 8 TCP connections on the path n_0—n_5 characterised by a 420 ms RTT. According to Eq 1 and 2, TCP with 256 bytes packet size and 420ms RTT should be as aggressive as our protocol (with the usual parameter setting). However, the simulation showed 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; in other words, given the same loss rate, our protocol sees a lower congestion-event rate than TCP does.

Figure 4 shows the same experiment performed in the real network. Using the testbed described above, we have run 4 in-

[2]The fairness index 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 (indicating that all instances get the same share) and $1/N$ (when one of the instances gets all the bandwidth).

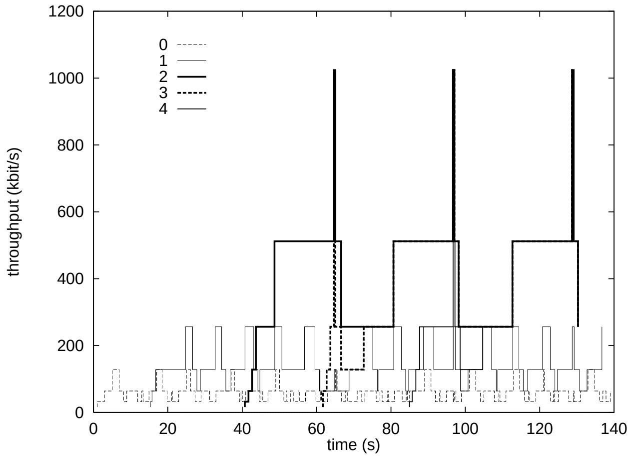

Fig. 5. The throughput for different receivers of the same protocol instance, starting at different times and behind different bottlenecks. The plot shows that receivers aggregate and stay in sync properly.

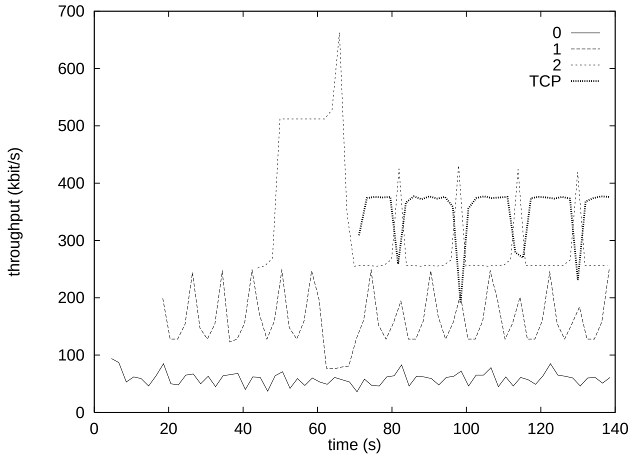

Fig. 6. Throughput for receivers on different bottlenecks with competing TCP traffic.

stances of our protocol and one TCP connection on our network path. Both in TCP case and in our case data rate has been evaluated at the application level averaging over non-overlapping windows 5 seconds wide. As we can see from the Figure, network load was almost entirely determined by our traffic: before we started the multicast transfers, the TCP connection was using all the available bottleneck bandwidth (excluding that used by non-responsive protocols); after our traffic started, the TCP throughput decreased, accounting for the bandwidth used by the multicast traffic. As we can see and according with Equations 1 and 2, with the parameters used (full-sized ethernet packets for TCP) and with with the RTT we had, our protocol was less aggressive than TCP.

F. Startup and steady state dynamics

A final set of experiments has been run to determine the dynamic behaviour of our algorithm. We were particularly concerned about the startup behaviour, to see how quickly a new receiver would synchronise with other receivers belonging to the same protocol instance and located behind the same bottleneck, and the steady state dynamics, to verify that receivers remain in synch and only move between adjacent layers.

On the topology of Figure 2,right, we have run an instance of our protocol placing a sender on node n_0 and five receivers on nodes n_5–n_9. Receivers on node n_5–n_7 have a bottleneck bandwidth of 1.5Mb/s while receivers on nodes n_8 and n_9 have a bottleneck bandwidth of .5Mb/s and .128Mb/s respectively. Moreover each bottleneck link is crossed by a random uncorrelated traffic which accounts for about half of the available bandwidth.

Figure 5 shows the result of the experiment: receivers on node n_5–n_7 (r_2–r_4) experience the same network conditions and converge reasonably fast to the same subscription level even if started at different times. They also keep synchronized until the end of the run. The other receivers, which stabilize on lower subscription levels, are not subject to any appreciable interference caused by the formers.

Figure 6 shows the throughput seen at the application level in a similar experiment. This time we have run receivers in n_2, n_3, n_4 together with a competing TCP connection on the path n_0—n_7 with 256 bytes packet-size. We can notice that TCP competes for the available bandwidth on the first bottleneck with receiver r_2 interfering very little with the other receivers and achieving a good bandwidth sharing.

V. Applications

The congestion control algorithm we have presented is applicable to both continuous data streams and to reliable bulk data transfer. In general, receivers will not receive all packets transmitted by the sender on all layers, either because their subscription level is too low, or because of losses due to congestion or to the hopping between layers. It should be noted, though, that our congestion control protocol aims at keeping the loss rate at very low values. Still, the effect of missing packets is obviously different for continuous data streams and for reliable bulk file transfers.

A. Continuous streams

In the transfer of continuous data streams, delay constraints generally do not allow to give hard guarantees on the integrity of the received data, and the recovery of lost packets is generally unnecessary. The goal, for a continuous data stream, is to transfer data with the maximum achievable quality for the available bandwidth. This is simply achieved by using the layered data organisation, where each subscription level corresponds to the same data transmitted with a different quality.

It is desirable to limit the latency in transferring data, and to make burst losses reasonably short. These are contrasting requirements since burst losses can be reduced by using techniques such as interleaving, at the price of an increase in the latency. Regarding latency, we should also note that the use of the sender-initiated probe produces a degree of burstiness in the traffic which requires at least τ_0 seconds of buffering at the receiving end in order to produce a uniform flow of data.

B. Reliable bulk data transfer

We come now to the problem of reliable bulk data transfer, which requires to find a proper layered data arrangement for a fixed-size data object, and an effective way for recovering from any packet that may have been lost because of congestion or during the jumps between different layers. We assume that the transmitter starts sending packets as soon as it receives a request from the first receiver, and continues transmission until there are active receivers. Transmission is always done on all layers, which are then selectively forwarded by the multicast routers. We also assume the existence of an upper level session management protocol which possibly collects feedback to decide how long to continue transmissions and select which packets to send.

The approach we follow is based on the use of FEC techniques (see [1], [12]). In principle, the problems of finding a suitable transmission order for the data, and achieving an effective recovery from losses, can be solved by passing all the k packets which make up the object to an encoder which *can* produce a large number $n \gg k$ of packets. The encoding is such that any subset of k encoded packets suffice to reconstruct the original data.

When the transmitter has to send a packet, it takes a new one produced by the encoder. Provided that each packets is not transmitted twice during the time a receiver needs to complete reception (the condition $n \gg k$ insures this), each receiver only has to collect k different packets, no matter which ones or which layer they come from, in order to complete reception. This approach has already been used by the authors in the RMDP protocol [13], a reliable multicast protocol which can be easily extended with the congestion control algorithm described in this paper.

The use of a FEC encoding with high redundancy makes all packets equivalent for data reconstruction, and solves many problems at once. First, there are no scalability problems due to the presence of uncorrelated losses at different receivers, since each receiver can conduct recovery autonomously, by simply waiting for more packets to come. Second, there is no need for detailed feedback from the receivers, since all received packets are equally good for data reconstruction purposes. Third, there is no need to devise a special data arrangement for the different layers in order to avoid that a receiver sees the same packet more than once.

The main drawback of FEC consists in the potentially high encoding and decoding costs. However, this overhead can be reduced by making use of a number of techniques:

- splitting files in blocks of small size, and applying FEC separately to each block;
- using FFT techniques [1] to compute the FEC packets;
- pre-computing the FEC packets at the server and storing them on disk;
- using probabilistic, linear-time algorithms such as those presented in [7] for the FEC encoding.

VI. Conclusions

We have presented a congestion control algorithm for multicast data transfer on the MBone, evaluated its performance, and shown its applicability to continuous stream and reliable bulk transfer. The algorithm does not require router support, or per-receiver status at the sender; it is completely receiver driver, and does not rely on any explicit protocol for receiver synchronisation or managing group membership. Analytical results, simulations and experiments on real networks have shown the algorithm to achieve a fair sharing of bottleneck bandwidth with other protocol instances and with TCP instances.

References

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

[2] 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.

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

[4] 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`

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

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

[7] M.Luby, M.Mitzenmacher, A.Shokrollahi, D.Spielman, and V.Stemann. "Practical loss-resilient codes", Proc. of the Twenty-Ninth Annual ACM Symp. on Theory of Computing, El Paso, Texas, 4-6 May 1997.

[8] M.Mathis, J.Semke, J.Mahdavi, T.Ott, "The Macroscopic Behavior of the TCP Congestion Aviodance Algorithm", CCR, Vol.27 N.3, July 1997.

[9] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", SIGCOMM'96, August 1996, Stanford, CA, pp.1-14.

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

[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.
Source code at `http://www.iet.unipi.it/~luigi/vdm.tgz`.

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

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

[15] N.Shacham, "Multipoint communication by hierarchically encoded data", Proc. of IEEE Infocom'92, (1992), pp.2107-2114.

[16] D.Taubman, A.Zakhor, "Multi-rate 3-D subband coding of video", IEEE Trans. on Image Processing 3, 5 (Sept.1994), pp.572-588.

[17] 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`.

[18] L.Vicisano, L.Rizzo, J.Crowcroft, "TCP-like congestion control for layered multicast data transfer", Research Note RN/97/67, UCL, July 1997. `http://www.cs.ucl.ac.uk/staff/L.Vicisano/rlc.ps`.