The RLC multicast congestion control algorithm

Luigi Rizzo, Lorenzo Vicisano, Jon Crowcroft

Abstract— **Decentralization of functionality is often a key ingredient in the development of scalable services. In this paper we present a fully distributed congestion control scheme for multicast data transfer in the Internet. Our Receiver driven, Layered Congestion control scheme (RLC) pushes all congestion control decisions to the receivers, leveraging on the capabilities of existing multicast routers to implement TCP-friendly congestion control in an extremely scalable way. RLC requires no per-receiver status at the sender, and no modifications to the multicast routing architecture.**

In this paper we present the RLC algorithm, define a simplified analytical model of its behaviour, and compare the analytical model with simulations and experiments in real networks. We also show the applicability of RLC to multimedia streams and reliable bulk data transfer, illustrating a number of applications and practical issues related to the implementation of RLC.

Keywords: congestion control, multicast, layered stream, reliable multicast, FEC.

I. Introduction

One of the key design principles of Internet protocols is that data sources make a considerate use of network resources, by adapting to available bandwidth and cooperating to achieve a fair sharing of resources among competing, well-behaved flows. This principle is at the basis of the TCP congestion control algorithms [7], which operate by reacting to congestion signals and adapting the data rate to the bandwidth available on the path between the sender and the receiver.

The deployment of multicast services on top of the Internet calls for similar schemes for use with multicast traffic. Because of the increasing use of multimedia tools for the distribution of audio and video on the Internet, there is in fact a growing use of potentially bandwidth intensive multicast streams, and the lack of proper congestion control mechanisms could easily render the network unusable, or force the extensive use of active bandwidth management techniques in the network elements to protect adaptive flows from non-adaptive ones.

However, congestion control is fundamentally more difficult for multicast than for unicast, even starting from the definition of the problem. The presence of multiple receivers, with potentially different bottleneck bandwidths to the source, makes it even hard to define a target data rate (or set of data rates) to be used in the communication. There is no single answer to the above problem: depending on the application, a multicast congestion control scheme might try to adapt to the slowest receiver, or drop some members from the set of receivers so that the remaining ones can support the minimum bandwidth requirements for the application, or finally select a set of data rates to adapt to the different requirements.

The next, and potentially more difficult issue, is to identify scalable and network-friendly mechanisms to estimate the target data rate(s) that can be used without causing congestion. Congestion control algorithms react to congestion *signals* that, depending on the type of network, can be either packet losses or explicit notification, and come either from the receiver or from the network.

Collecting multicast feedback is a difficult process: if done too slowly, it might render the feedback totally useless; if done too aggressively, can cause what is called "feedback implosion", consisting in an unordinate amount of feedback directed to the source. Many different approaches have been used to solve this problem, generally based on randomization and suppression techniques, election of representatives, or construction of hierarchies for feedback aggregation. In particular, the latter two are just examples of a design principle – decentralization of functionality – which is often the key to the development of scalable services.

The Receiver-driven, Layered Congestion control algorithm (RLC) presented in this paper makes extensive use of this principle in order to achieve a very high scalability. RLC is designed to support one-to-many communication to potentially large sets of receivers with different bandwidth requirements. Both continuous streams and reliable data transfers are supported by RLC, as it will be shown in Section V. RLC uses a hierarchical, layered scheme for data transmission, where receivers can join to one or more multicast groups to receive data at a rate approximately matching their bandwidth to the source – this translates into different quality levels in the case of multimedia streams, or in faster transfer times for reliable data communication.

Scalability in RLC comes from full decentralization of functionality: each receiver takes congestion control decisions autonomously; traffic filtering is implemented by routers on the multicast distribution tree; and a no-overhead scheme is used to synchronise receivers behind the same bottleneck (and belonging to the same protocol instance), so that an effective use of the bandwidth is achieved. No explicit forms of group membership are used, so that the algorithm is fully scalable and simple to implement, both at the sender and the receiver side.

Luigi Rizzo (luigi@iet.unipi.it) is with the Dipartimento di Ingegneria dell'Informazione, Univ. di Pisa; Lorenzo Vicisano (lorenzo@cisco.com) is with CISCO Systems, San Jose, CA; Jon Crowcroft (jon.crowcroft@cs.ucl.ac.uk) is with the Department of Computer Science, University College, London. This work was performed while Lorenzo Vicisano was with University College London, and has been partially supported by the Italian MURST in the framework of Project MOSAICO, and Esprit LTR Projects 20422 - MOBY-DICK and 21926 – HIPPARCH.

A. Related work

Layered data organization had been used in related work [1], [5], [10], [8] to implement some form of adaptive multicast data distribution using two or more multicast groups. These works mainly focus on video distribution, also exploiting research results on layered video encoders e.g. [18]. To our knowledge, the applicability of layered techniques to reliable multicast bulk data transfer was first suggested by the authors in [20], by making use of FEC techniques [15]. The same approach has been subsequently used by others in conjunction with different FEC encoders [2].

The Receiver-driven Layered Multicast scheme (RLM) by McCanne et.al [10] is the first work that tries to use a layer hopping algorithm to adapt to the available bandwidth, through the use of a state machine driven by timeouts and loss measurements. RLM does not attempt to compare its behaviour with TCP, nor provides an analytical model of its behaviour. LVMR [8] has similar goals but resorts on a hierarchical architecture to reduce the amount of state at receivers and to speed up the response to changes in network conditions.

In a recent work, Rubenstein et al. [17] formalized heterogeneous (multi-rate) congestion control as far as fairness and bandwidth utilization are concerned. They described how layered multicast transmission approximate the ideal multi-rate regulation and found out that a good inter-receiver synchronization (see section III-A.2) is the key requirement for achieving an efficient utilization of network bandwidth. They also showed that the RLC synchronization algorithm is able to address this issue with minimal discrepancy from the ideal case.

RLC was originally presented in [20], introducing a number of novel features such as fair competition with TCP flows, the achievement of receiver synchronisation without the use of a dedicated group membership protocols or explicit message exchange among receivers, a very simple protocol structure both at the sender and the receivers, and showing how layered techniques could be applied to reliable bulk data dissemination. The simplicity of the protocol also allowed us to develop an analytical model that matches very closely simulations and experimental results.

The paper is structured as follows. In Section II we briefly describe the basic mechanisms used by RLC, namely the loss response function, multicast group membership, and layered data organization. RLC is presented in detail in Section III, while Section IV contains the performance analysis, first using a simplified steady state model, and then validating it with simulations and experiments on real networks. Applications of RLC are discussed in Section V, while final comments are presented in Section VI.

II. Basic mechanisms

RLC is based on a target response function implemented through two basic mechanisms that make the algorithm effective and easy to deploy. The response function is the TCP equivalence formula; the basic mechanisms are the management of multicast membership – already in place in the Internet architecture – and a layered data organization – implemented at the end nodes. These two mechanisms are available without requiring any modification to the current Internet service, thus allowing for an immediate deployment of RLC. In addition to this, the usage TCP response function determines a long term response to losses that can easily be tuned in order to make RLC coexist with background TCP traffic, preventing that one of them starves the other. In this section we review these three points.

A. Congestion response function of TCP

Congestion control algorithms react to congestion signals (or their absence) by decreasing or increasing the transmit window (or rate) seeking for an equilibrium point. Because IP networks do not have an explicit congestion notification mechanism[1], TCP interprets the loss of one or more packets at the receiver as a congestion signal, and the successful delivery of packets as a signal of no congestion.

The steady state behaviour of TCP congestion control has been studied in detail by several authors [9], [12], resulting in more or less complete analytical models of TCP which indicate the relation between the packet loss rate p and the throughput R of a TCP connection depending on various parameters. One such model [9] is described by the "simple equation" below:

$$R = \frac{s \cdot c}{RTT\sqrt{p}} \qquad (1)$$

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. This model holds for small values of p, when losses can be recovered without incurring in timeouts. At high loss rates, TCP tends to lose its ACK-based self clock, and the actual throughput has a sharp decrease and becomes highly dependent on the length of the retransmission timeouts. More complex models [12] have been developed to cover such extended range of loss probabilities.

The goal of a TCP-friendly protocol (be it for unicast or multicast) is to mimic, to some degree, the response of TCP to congestion signals. This behaviour can be either intrinsic of the protocol (e.g. in the case of TCP, the simple equation is a result of the structure of the protocol) or a design goal (as we will see, in the design of RLC we start from a desired behaviour to set the operating parameters of the congestion control algorithm).

We should keep in mind that 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 notice and react to a congestion signal.

[1] Recent proposals have been published [13] to support Explicit Congestion Notification; TCP and RLC can easily make use of this kind of congestion notification if available.

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 [4]. Packets for a given multicast group are only forwarded into those network segments leading to active receivers (either end nodes or multicast routers serving other active receivers) for that group.

New receivers willing to join a group send 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). This *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).

When receivers are not interested in a group anymore, they inform the router with a different IGMP message. 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, this *leave* phase can take a considerable time (a few seconds) to complete, 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 on the flow of data in a subtree, although the minimum duration of the on period is constrained by the leave delay.

C. Layered organisation of data

RLC is designed to support the distribution of the same data (possibly, with different quality) to a set of receivers with different, increasing bandwidths B_i's, $i = 0 \ldots l-1$. Receivers on the distribution tree might have different bottlenecks (hence require different bandwidths $B_i \neq B_j$), so a possible solution to this problem is to distribute data using a cumulative, layered data organisation based on l multicast groups or *layers*. Each layer carries data with bandwidth L_i, in such a way that $B_i = \bigcup_{j=0}^{j=i} L_j$.

With this organization, each receiver can tune its receive bandwidth by joining the appropriate number of layers, and there is no duplication of data if receivers with different requirements share the same links.

The different data rates can be used on two different dimensions, depending on the type of data being transferred. For streaming data such as audio and video, or other volatile information, each additional layer can be viewed as a refinement of the information available in the previous layers, e.g. conveying higher resolution or frame rate video refinements, or additional audio channels, and so on. In the case of reliable data transfer, the additional layers can be used to reduce the overall transfer time. In both cases, we can think of a "quality" index associated with each layer set.

Defining a layered data organisation is essentially a source coding problem, and finding a suitable data arrangement for the various layers depends entirely on the application domain. In principle, the goodness of a layered organization can be evaluated by looking at how the quality index increases with B_i, and ideally we would like to see a linear quality increase. In practice, though, there are other design constraints that affect the goodness of the packet scheduling policy. One of these is that receivers are allowed to hop through different layers depending on the instantaneous bandwidth availability.

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* (number of layers they listen to) which best matches the available bandwidth. Using a limited number of layers also makes RLC's operation a more acceptable burden to the multicast routing protocols, by reducing the frequency of join an leave requests.

III. Receiver-driven Layered Congestion control (RLC)

A. RLC components

RLC takes advantages of the mechanism described in the previous Section by letting receivers hop between layers depending on congestion signals. The algorithm is made of four components:

- a layer hopping rule, which defines the basic behaviour of receivers in response to congestion signals;
- synchronization points, which help synchronize the behaviour of receivers behind the same bottleneck;
- sender-initiated probes, which reduce the number of errors in estimating the available bandwidth;
- a deaf period, which is used to circumvent the slow response of IGMP to group leave messages;

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 scales poorly, since different groups of receivers with different requirements may exist, and adapting to the needs of one set of receivers may penalize others with contrasting requirements.

As in other proposals appeared in the literature such as RLM [10], we achieve the desired effect by using the multicast routers driving bottleneck links to modulate the sending rate for that 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 this Section we ignore the problem of recovering missing packets (generally lost due to congestion), as it is orthogonal to congestion control and there are application-specific solutions to this problem.

A.1 Layer hopping

RLC 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. In principle, and similarly to RLM, an RLC 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.

As it will be shown in Section IV, with a proper choice of bandwidths B_i's and delays $t_p(i)$'s (e.g. doubling B and t_p at each layer), the above algorithm yields a relation between throughput and loss rate similar to that of TCP, namely $T \propto 1/\sqrt{p}$.

In practice, the simple algorithm presented above must be refined to deal effectively with the presence of multiple receivers, and with the peculiarities of multicast IP.

First, multiple receivers can completely defeat the operation of the algorithm if they act in uncoordinated ways, thus causing persistent congestion on the bottleneck links and even worse causing receivers to lose perception of the effect of their joining/leaving layers.

Second, the length of the leave delay makes *failed* join 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.2 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) dropping a layer (leaving a group) has no effect unless all receivers sharing the same 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;

Minimizing these problems requires coordinating join attempts from different receivers, which otherwise would tend to diverge because of different start times, loss patterns, and the history of events experienced by each of them[2]. RLC uses 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.

We want to remark the simplicity of using synchronization points instead of an explicit protocol for receiver coordination, since the latter could pose scalability problems and be slow to converge.

A.3 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 long-lasting consequences. 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.

RLC includes 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 transmission rate in all the layers is increased so that each subscription level carries the data rate carried by the upper level in normal conditions. 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 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

[2] Note that skews in propagation delay among receivers do not affect the convergence of the protocol as the relative ordering of all the events is respected anyways.

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.

A.4 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 leave the multicast router sufficient time to respond to the leave request.

In recent work [16] we have addressed the problem of slow IGMP response, and proposed a minor modification to IGMP (router side only) which renders leave response instantaneous making use of prediction techniques. While this modification would improve the behaviour of RLC and similar protocols when dropping layers, it requires software modifications to the leaf multicast routers, and so it will require longer times for deployment.

B. *The full algorithm*

We are now ready to discuss the full RLC algorithm. We assume that l different bandwidths are offered, $B_i, i = 0 \ldots l-1$, with increasing values. 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 simplicity reasons, in the remaining of the paper we will assume that data rate is exponentially spaced between a subscription level and the subsequent by a factor of 2 ($B_{i+1} = 2B_i$). RLC can in principle work with any increasing distribution, however using an exponential distribution facilitates some protocol design tasks, such as the placement of synchronization points (see below). Note also that there are application and network level considerations that suggest to use exponential distribution (see Section II-C).

B.1 RLC Transmitter

For each bandwidth B_i packets are normally transmitted at a constant rate, appropriately placed onto the different layers. The nominal inter-packet time of a layer set halves as the subscription level i increases.

Synchronisation points (SP's) correspond to specially-tagged packets, and are spaced proportionally to the bandwidth corresponding to the subscription level (i.e., exponentially spaced). SP's for level $i+1$ are a subset of those for level i. We call t_0 the distance between SP's at level 0.

Bursts of packets (sender probes) are sent right before SP's, and are generated by sending pairs of packets at each transmission time, followed by "holes" (i.e. no transmissions) for a number of slots to compensate the excess transmissions. The size of the burst is also proportional to the bandwidths, starting from a burst of 2 packets at level 0 and doubling each time.

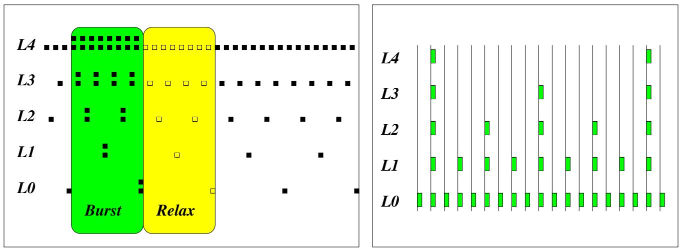

Fig. 1. Left: the sequence of packet transmissions; right: the location of synchronisation points at the various layers. $W = 8$ and five layers.

Additional parameters that appear in the protocol are the packet size, s and the number of packets between SP's at level 0, W, which are related by $W = B_0 t_0 / s$. As a reference, Figure 1 shows the sequence of packet transmissions and the location of SP's for five layers (and $W = 8$).

We note that an RLC transmitter is remarkably simple, basically just a timer process that determines packet transmissions on the various layers, and it is completely outside of the congestion control loop, which is really made by receivers and multicast routers on the multicast distribution tree. Among other things, this means that we can feed a multicast subtree with a streaming source by just making use of a transcoding process [6] and still take advantage of congestion control; or, we can record at the source the packets generated on the various layers by an RLC transmitter, and just replay them to generate a replica of the transfer adapting to different network conditions[3].

B.2 RLC Receiver

Subscription levels can increase only at SP's, and decrease at any time, using the following rules (we use the term "congestion" instead of "packet loss" as the algorithm responds to whatever congestion signal the network can report, be them dropped packets or ECN bit set in the packet header):

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

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

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 RLC are the use of synchronisation points and the rules for changing

[3]We in fact did this in one experiments with a reliable multicast application (RMDP/RLC) which sends data encapsulated in RTP packets. The source was recorded with `rtpdump`, and later the recording was played with `rtpplay` on the same number of layers to re-run a congestion-controlled reliable file transfer on a different bottleneck.

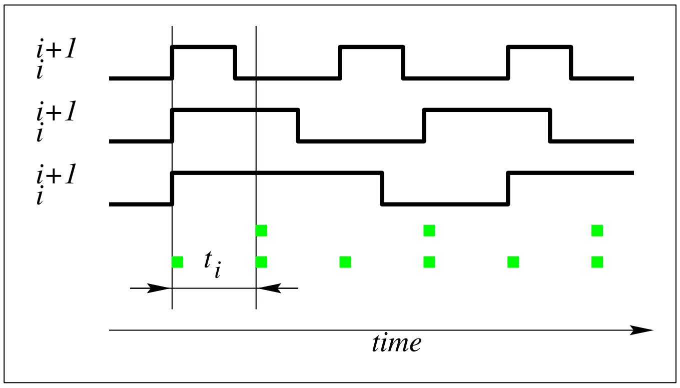

Fig. 2. Some layer hopping patters for our simplified analytical model.i Squares on the bottom represent the location of synchronization points at levels i and $i+1$.

subscription levels, which make the algorithm considerably simple and very scalable. It is also worth mentioning that RLC employs no receivers' feedback directed to the source – as in RLM, feedback is implicit in the form of IGMP messages to the local router which performs feedback aggregation and forwarding.

IV. Performance analysis

In this section we evaluate the performance of our congestion control algorithm. We first derive an analytical relation between throughput and loss rate basing on an approximate steady state model. The analytical model is then validated through simulations and measurement on the Internet, which show a very close match to the analytical value. Due to lack of space, here we only report the main results of simulations and experiments; full details are reported in [20].

A. Analytical model

To compute the relation between throughput and loss rate, we use a simplified steady-state model of the algorithm. We assume that receivers oscillate between subscription levels $i+1$ and i (see Figure 2), dropping layer L_{i+1} at the first loss (which we assume to be the only one in a cycle), and joining L_{i+1} at the first useful synchronization point (i.e. after a full interval without losses at level i). Remember that both rates B_i and the distance between synchronization points t_i are exponentially spaced, i.e. $B_i = 2^i B_0$ and $t_i = 2^i t_0$, s is the packet size, and $W = B_0 t_0 / s$ is the number of packets between SP's at level 0.

Within a cycle, a time αt_i will be spent at level $i+1$, whereas the full cycle will last

$$T = 2^i t_0 \lfloor 2 + \alpha \rfloor$$

Since in our model we assume one lost packet per cycle, the loss rate is $p = 1/N$, where N is the total number of packets received in a cycle. We can easily compute N by considering the time spent at each level (and the rates B_{i+1} and B_i), and we obtain

$$1/p = N = (2^i)^2 W (2 + \alpha + \lfloor \alpha \rfloor)$$

p mainly influence the subscription level i, so we use the previous equation to express i as a function of p:

$$2^i = \frac{1}{\sqrt{pW(2+\alpha+\lfloor\alpha\rfloor)}} \qquad (2)$$

At this point we are ready to compute the average throughput R of a session $R = sN/T$. By replacing N and T with the above equations, and replacing the terms 2^i with Eq. 2, we obtain:

$$R = \frac{s}{t_0}\sqrt{\frac{W}{p}}c'(\alpha) \qquad (3)$$

which has the same dependency on p as the simple TCP equation, and where $c'(\alpha)$ depends very weakly on α.

In the analytical model we have assumption a single loss per cycle, because even if there are more (and this is likely to happen, because of the IGMP leave delay), RLC will react only once for stability reasons. As a consequence, when matching experimental data (from simulations and measurements in real networks) to the analytical model, we have used the burst loss rate as a parameter, computed by considering closely spaced losses as a single one. With this correction, experimental results fit well our model.

B. Parameter tuning

RLC's response (both steady state and dynamic) can be tuned by acting on the parameters appearing in Eq. 3, so as to achieve the desired response to congestion in terms of dynamic response and aggressivity.

t_0 directly influences the speed of response of RLC to improvements in network conditions, but it cannot be set too low because its purpose would be defeated by the leave delay and the subsequent deaf period. A reasonable value for t_0 is in the order of 1 s. Compared with TCP (which reacts in the order of 1 RTT), this means that RLC can be slower detecting the availability of additional bandwidth.

B_0 is the bandwidth used at the lowest layer. 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 need of providing enough packets between synchronization points at the lowest layer to implement the burst, relaxation and normal transmission periods. This parameter is related to B_0 as the amount of data transmitted between SP's at the lowest layer is $B_0 t_0$ bytes, which we split in W packets of size s each. W must be at least 2 (but hopefully larger), and 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 $t_0 = 1s$, and chosen $s = 256$ bytes and $W = 8$, which result in $B_0 = 2$ KBytes/s.

C. Comparison of RLC and TCP congestion response

Although RLC tries to mimic the behaviour of TCP congestion control, and Eq. 1 and 3 only differ by a multiplying constant, there are some differences between the behavior of the two algorithms.

The main difference between RLC and TCP response functions is the lack, in RLC, of an explicit dependence on the Round Trip Time of the connection. There is an economic argument in favour of the RTT dependence of TCP connections, stating that biasing throughput against RTT introduce a desirable penalty in TCP connections that use large amount of network resources. Even though this argument is only valid in the assumption that RTT is a good measure of the resource utilization, the experience accrued in several years of TCP deployment does not seem to deny its validity.

The case of heterogeneous multicast rate regulation is quite different in this respect. First of all it is not clear which RTT should be used, as the rate regulation can happen at any point inside the network and a given sender-receiver path is not necessarily crossed by the same session rate. Second, if we applied economic considerations, we should take into account that a link leading to multiple receivers produces a better service than a link leading to a single receivers. A large number of receivers sharing a link can compensate for the number of links crossed (or the distance) and be the dominant factor.This suggests that, in the case of multicast, the issue of resource utilization is more complex than in unicast and that the RTT approximation can be too simplistic.

A second difference is that our protocol works by controlling the data rate, while TCP congestion control is based on a window that allows for the 'conservation of packets' principle [7]: 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 acknowledgements to work, making TCP break at high loss rate, as discussed below.

Another important difference is that our algorithm 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 might obtain a worse bandwidth utilization at low levels of statistical multiplexing; the other one is that the minimum allowed throughput can be higher than what the network can sustain; this happens in presence of high losses. Both Eq. 1 and 3 are only valid at low loss levels, corresponding to a TCP session able to recover through fast retransmit, and our algorithm used with a bandwidth $B \geq B_0$. If these conditions do not hold, the RLC source becomes not controllable, since we do not have a way to further reduce the sending rate. But because B_0 is mainly dictated by the application, we believe that it is the application level that should react to heavy congestion at layer 0, possibly by permanently disconnecting from the session.

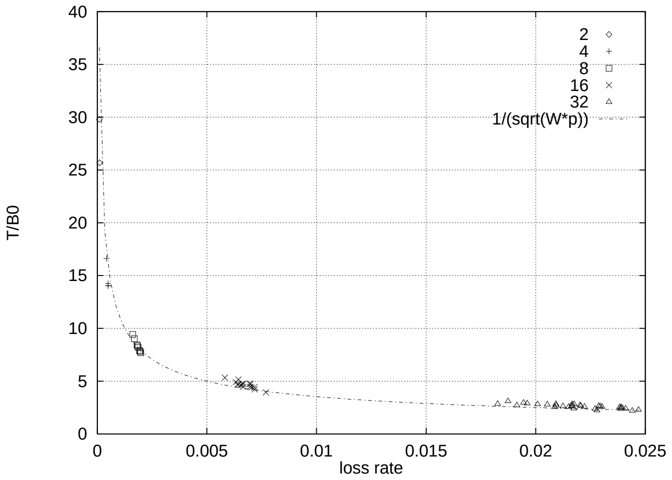

Fig. 3. The relation between throughput and (burst) loss rate in simulations with RED routers, compared to the analytical results from Eq. 3 (solid line).

D. Validation of the model

To validate our analytical model, we have both simulated the behaviour of several instances of the protocol sharing a bottleneck link, and run RLC instances on real Internet paths. More details on the simulations and experiment setups and results are presented in [20].

Simulations have been performed using the `ns` network simulator [19] (an RLC implementation is also present in recent `ns` distributions), with both RED and drop-tail routers. The graphs presented in this paper refer to simulations with N sender/receiver pairs spread on two paths with different delays sharing the same bottleneck. Senders were started at random times over an interval of $N/2$ seconds, and receivers had been kept active for 300 seconds.

In Figure 3 we compare Eq. 3 with simulation results in the case of RED routers: as we can see there is a very close match between the model and simulations. This is because RED routers send congestion signals before the queue saturates thus matching well our assumption of one loss event per cycle. Simulations with drop-tail routers (also confirmed by experiments) show a slightly higher throughput. This can be explained as drop-tail routers tend to provide a delayed congestion notification (packet drop) followed by more concentrated loss events, which will be partly ignored by RLC as they fall within the deaf period.

A similar experiment has been done using an implementation of RLC on a real Internet path between UCL (London) and DEIT (Pisa). In order to compete with congestion controlled traffic, we setup a multicast tunnel (without rate limitations) between the multicast routers feeding the two subnets, with multiple senders on four hosts at one site, and multiple receivers two hosts at the other site. Figure 4 shows the comparison between experimental and the model. The actual throughput, slightly higher than the analytical value, can be motivated by the (probable) use of drop-tail routers in the network, whose effect has been discussed above.

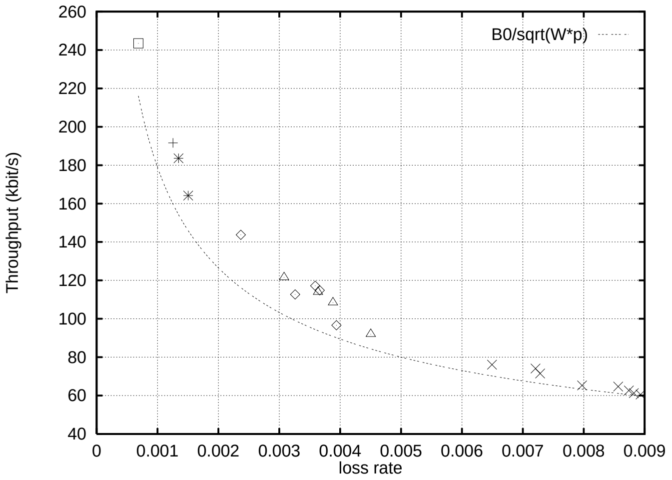

Fig. 4. The relation between throughput and (burst) loss rate in real measurement, compared to the analytical results from Eq. 3. Protocol instances run concurrently are represented by the same symbol.

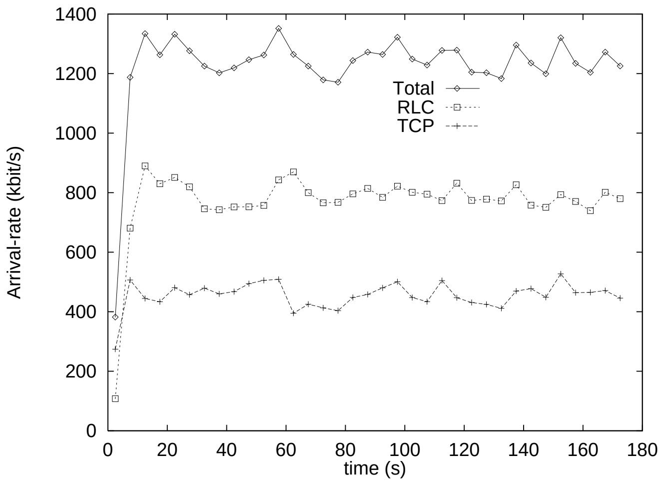

Fig. 5. Aggregate throughput of TCP connections and that of our protocol instances.

In [20] we also measured the min/max fairness index [3], which result in values very close to unity for RED routers, and only marginally lower for drop-tail routers and few protocol instances (a situation where the greater burstiness of drop-tail losses slightly deteriorates receiver's response and is the source of some minor unfairness).

E. Fairness with TCP

Interactions with TCP have been investigated with a second set of simulations and experiments. Using the `ns` simulator, we have run 8 multicast protocol instances as in the previous experiments, together with 8 TCP connections on a path sharing the same bottleneck and characterised by a 420 ms RTT (which, according to Eq 1 and 3 should yield the same throughput as RLC instances for the same packet loss rate).

Figure 5 plots the aggregate throughput of TCP connections and that of our protocol instances. The graph clearly

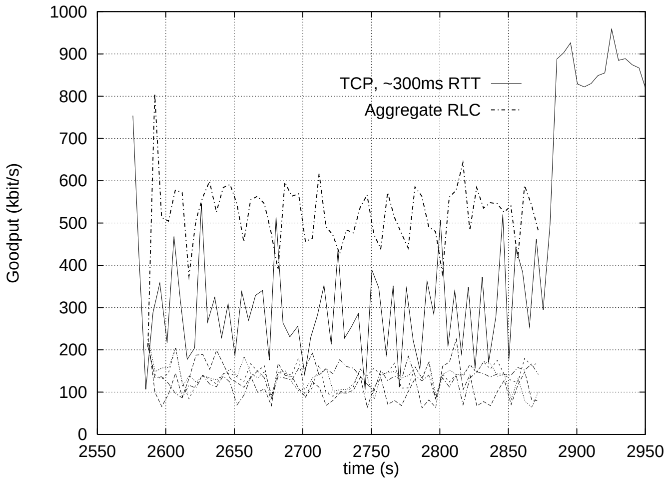

Fig. 6. Bandwidth sharing among one TCP connection (top) and 4 protocol instances (bottom) on a real network. $RTT \simeq 300$ms, and TCP packet size of 1440 bytes.

shows that RLC is slightly more aggressive than TCP, but i) this was expected as RLC considers closely spaced losses as a single event, whereas TCP does not, and ii) is not a concern since it is known that TCP throughput depends on the RTT which is not under our control, so a session with shorter RTT will be more aggressive than RLC and vice-versa. What is really more important is that TCP and RLC do not starve each other when competing – and this was clearly shown both by simulations and by experiments. As an example, Figure 6 shows the behaviour of TCP and RLC in a similar experiment run on the real network. Using the testbed described above, we have run one TCP connection, and after TCP settled we started 4 RLC instances. 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 the graph shows, both TCP and RLC connections share the available bandwidth without starving each other.

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.

For this kind of experiments we have used a single source and multiple receivers spread on a topology comprising cascaded bottlenecks and joining the session at different times, while non congestion-controlled background traffic was consuming half of the available bandwidth. Also in this case experiments have shown that receivers are able to synchronize within a few synchronization points and operate in synch afterwards (see [20] for full details).

V. Applications

The congestion control algorithm we have presented is applicable to both continuous data streams and to reliable bulk data transfer. In fact, we have developed RLC as a library and are using it in a few applications such as a multicast reliable file transfer application, a multicast slide show application, and a layered video transcoder[4].

RLC aims at selecting for each receiver the subscription level that minimizes the loss rate, thus keeping it at low values. However, some losses are generally unavoidable as part of the feedback mechanism. Although the recovery process is orthogonal to congestion control, it is worth mentioning how RLC can be used with different types of data streams.

A. Continuous streams

In the transfer of continuous data streams such as audio or video, delay constraints generally do not allow the use of retransmission schemes to recover lost packets. Single-channel multicast applications generally use one of the following strategies:

- ignore the problem, if the data stream (e.g. video) can tolerate high losses and still provide a reasonable quality;
- protect the data stream by using forward error correction techniques, implemented with standard FEC codes, or with lower-quality replicas of the data stream.

As RLC essentially implements a selective dropping scheme in the set of layers, we can organize data on the various layers by putting the most important information on the lowest layers, and using the higher ones for refinement information. Several layered coders [11] or transcoders [6] have been proposed in the literature, typically for video streams, and their integration with RLC is straightforward. As a further refinement, having characterized the behaviour of RLC, a source can know the mean loss rate corresponding to the various subscription levels, and can protect the data stream with variable redundancy depending on the layer.

B. Reliable bulk data transfer

For reliable data transfer, the recovery of all missing packets is a fundamental issue. Recent research has shown as FEC-based techniques are a very effective and scalable way to implement loss recovery in multicast communication using a single multicast group. Such techniques are based on the use of an encoder [14] capable of producing n different encoded packets starting from $k < n$ source packets. The encoded packets have the property that any subset of k (or slightly more, in the case of a probabilistic encoder) packets enables the reconstruction of the original. In single-group reliable multicast applications such as RMDP [15], the encoded packets are transmitted either on demand or unconditionally until receivers have accumulated enough packets to reconstruct the original data.

The application of this principle to layered data transfer is in principle straightforward: we can just make use of an encoder with $n \gg k$ and generate a new packet each time a transmission is scheduled on each layer. Receivers will then be able to hop between layers depending on available bandwidth without concern on which packets get lost – the only constraint being to accumulate a sufficient number of packets to perform reconstruction. The condition $n \gg k$ guarantees that the same packet will be retransmitted only after a long time, hopefully long enough to avoid that receivers complete reconstruction without receiving the same packet twice or more (something that would be useless to the purposes of data reconstruction).

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 use of FEC has a drawback, though, which consists in the potentially high encoding and decoding costs, which however are perfectly bearable with today's hardware. Even though conventional FEC encoders/decoders such as the one shown in [14] require $O(k)$ operations per packet produced, the multiplying constants are extremely small, and encoding speeds in the Mbit/s (or higher) can be achieved on commodity hardware with values of k in the range 32..64. For long files, where the use of a large k would be impractical, encoding costs can be reduced by either splitting the file in a number of bounded-size blocks and applying the FEC encoding separately to each block, as done by the authors in [15], or using probabilistic but asymptotically better FEC encoders as those presented in [2].

VI. Concluding remarks

We have presented RLC, a novel congestion control algorithm for multicast data transfer on the MBone, supplied an analytical model of its behaviour, and validated the model with simulations and experiments in real networks. The main objectives in the design of RLC were good coexistence with TCP, scalability, ease of deployment. We believe we fulfilled our goals: the analysis and experiments presented in this paper all show that the protocol behaves as desired; scalability comes from the extreme simplicity of the protocol and its fully distributed nature; and RLC is designed to make use of existing network infrastructure and coding techniques to ease its deployment both for multimedia streams and reliable bulk data transfer. We have built RLC as a portable library in C and Java, and are currently implementing a set of applications using it for both multimedia streams and reliable data distribution.

References

Note for the reviewers: IEEE Network directions require references to be numbered in order of citation in the paper, recommends a number of about 10 references for the final paper, so this list of references will need to be trimmed accordingly for

[4] Source code for all these applications is available from the authors.

the final version of this paper.
In this draft we leave the complete list of references in alphabetical order to ease the review process.

[1] T.Brown, S.Sazzad, C.Schroeder, P.Cantrell, J,Gibson, "Packet video for heterogeneous networks using CU-SeeMe", IEEE Intl. Conf. on Image Processing, Lausanne, Sept.96

[2] J.W. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", ACM SIGCOMM'98, Vancouver, CA, September 1998.

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

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

[5] D.Hoffman, M.Speer, "Hierarchical video distribution over Internet-style networks", IEEE Intl. Conf. on Image Processing, Lausanne, Sept.1996

[6] G.Iannaccone, L.Rizzo, "A Layered Video Transcoder for Videoconference Applications", Mosaico Technical Report PI-DII/4/1999, Univ. di Pisa, July 1999. (`http://www.iet.unipi.it/~luigi/lvt.ps.gz`)

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

[8] X. Li, S. Paul, P. Pancha and M. Ammar, "Layered Video Multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rating Control", IEEE Infocom '98, Mar. 1998.

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

[10] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast", SIGCOMM'96, August 1996, Stanford, CA, pp.1-14. (`ftp://ftp.ee.lbl.gov/papers/mccanne-sigcomm96.ps.gz`)

[11] S. McCanne, M. Vetterli, "Low-complexity Video Coding for Receiver-driven Layered Multicast", IEEE Journal on Selected Areas in Communications, vol. 16, no. 6, pp. 983-1001, Aug. 1997.

[12] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM SIGCOMM'98, Vancouver, CA, September 1998.

[13] K.Ramakrishnan, S.Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC2481, Jan. 1999.

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

[15] L.Rizzo, L.Vicisano, "RMDP: an FEC-based Reliable Multicast protocol for wireless environments", ACM Mobile Computing and Communications Review, Vol.2, n.2, April 1998, pp.23-31.

[16] L.Rizzo, "Fast Group Management in IGMP", Proc. of the 1998 Hipparch Workshop, London, June 1998 pp.32-41.

[17] D. Rubenstein, J. Kurose, D. Towsley, "The Impact of Multicast Layering on Network Fairness", To appear in ACM SIGCOMM '99. `http://www-net.cs.umass.edu/~drubenst/publish/Rubenst99_MultiFair.ps`.

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

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

[20] L.Vicisano, L.Rizzo, J.Crowcroft, "TCP-like Congestion Control for Layered Multicast Data Transfer", IEEE Infocom'98, San Francisco, CA, Mar.28-Apr.1 1998.