

# Pruning Algorithms for Multicast Flow Control

Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley, Haifeng Zhu

Sun Microsystems Laboratories

Burlington MA 01803

01.781.442.0525

{dahming.chiu, miriam.kadansky, joe.provino, joseph.wesley} @sun.com,  
haifengz@ece.cmu.edu

## ABSTRACT

Multicast flow (congestion) control establishes a data rate for a multicast session based on prevailing bandwidth availability given other network traffic. This rate is dictated by the sender's path to the slowest receiver. This level of performance, however, may not be in the best interest for the multicast group as a whole. A pruning algorithm is then used to identify and remove some slow members so that the performance is acceptable for the whole group. This paper discusses the conceptual issues with pruning, and proposes practical algorithms for pruning. The crux of the problem is to achieve a balance between speed and accuracy, because increased accuracy tends to require monitoring for a longer time and using more global information. We evaluate and compare different strategies using both simulation and measurement of real implementations.

## General Terms

Algorithms, Performance, Reliability, Experimentation.

## Keywords

Multicast, group utility, flow control, pruning, performance.

## 1. INTRODUCTION

A multicast flow may have many receivers. The job of the multicast flow control algorithm is to find the suitable rate for the slowest receiver, and use that as the transmission rate for the whole multicast group.

As the multicast group becomes larger and more heterogeneous, the rate for the slowest receiver may prove to be too slow for the benefit of the whole group. In such scenarios, it is appropriate to prune the slowest members from the multicast group so that the resulting multicast flow can operate at a satisfactory rate.

While there are many multicast flow control algorithm proposals, there has been relatively little discussion of how pruning of slow members works. In this paper, we examine various pruning algorithms in a general light so that the concepts can be applied to different flow control scenarios.

## 2. RELATED WORK

In multicast routing protocols, a common technique to maintain a multicast tree is *flood-and-prune*, which is used in the Distance Vector Multicast Routing Protocol [20]. In this context, the interior node (router) sends data down all links (except the incoming link) as if it were broadcast data. If a downstream node does not need the data, it sends a prune message back to remove itself from the distribution tree. This algorithm is loosely related to our work in the sense that flood-and-prune maintains a distribution tree, whereas our algorithms maintain a distribution tree operating at a certain data rate (or rate range).

One elegant approach to deal with multicast receivers with different receive rates is proposed in the context of audio/video multicast applications. In this case, the data can be encoded as multiple layers. The base layer provides the information with minimally acceptable fidelity, and each additional layer provides improvements to the fidelity. The source multicasts the different layers using different multicast addresses simultaneously. The receivers adaptively determine the number of layers they receive based on their receive rates. The receiver adaptation (deciding whether to join a particular layer) is related to the pruning algorithms that we discuss later on in this paper. This idea and variations of it have been published in many papers, among them [19] and [16]. The problem discussed in this paper, however, is for a single layer multicast flow. Therefore, the service delivered to each member is binary: either *pruned* (no service) or *not pruned* (full service).

The pruning (they used the term "discarding") of slow receivers problem is briefly discussed in [4] and [18], as one of three problems associated with multicast congestion control. While the papers surveyed many studies and proposals for multicast flow and congestion control, the discussion on the pruning problem is very brief.

In [15], the pruning (they used the term "ejecting") of slow receivers is studied in the context of the FRM (Fully Reliable Multicast) algorithm<sup>1</sup>. Some of the techniques of selecting slow

---

<sup>1</sup> A reviewer pointed us to [15].

receivers in [15] are similar to what are reported in this paper, but the discussion is very brief and does not cover many of the issues when integrating with a real protocol.

The work reported in [9], [11] and [10] is also related to our problem at hand.<sup>2</sup> Jiang et al developed a notion of inter-receiver fairness (IRF), to reflect the receivers' collective utility function. Based on this IRF, the receivers are then divided into several groups, each covering a different range of receive rates. The slowest group can be considered as the group of receivers pruned from the session. The sender then replicates its transmission into one stream per subgroup (except the pruned group). The data rate for each subgroup is determined using the IRF function as well.

Although we are trying to address the same issue, our approach is quite different from that proposed by Jiang et al. We summarize the differences here:

1. First we show that the receiver utility function can take a more general form than the proposed IRF (a weighted sum of each receiver's utility); on the other hand, we also believe it is only practical to use a very simple criterion for pruning (by setting a minimum data rate). The assumption in JAZ that the receiver rates can be measured applies only to ATM networks and not to the Internet. This means applying the IRF dynamically (to divide receivers into subgroups and determine the subgroup rates) is not practical for the Internet.
2. Secondly, we are not convinced replicating the multicast stream into multiple (simultaneous) streams is necessarily a good idea as it increases the bandwidth consumption. If receivers were divided into multiple groups, then a multi-layered approach would be more attractive.
3. Thirdly, the use of an IRF-determined rate rather than the rate of the slowest member of a subgroup is not TCP-friendly. This deviation from TCP-friendliness should be quantifiable, hence is not necessarily a big problem.

When we present our results, some of these points will be revisited in more detail.

Finally, in the context of some studies of the economics of sharing multicast costs ([8], [14]), one possible result is to not forward multicast data along certain multicast links when the cost exceeds the utility to the downstream receivers. This is conceptually similar to our idea of pruning. This body of work is more focused on the economics and computational complexity aspects of a seemingly related problem, whereas our work addresses the practical and engineering aspects of pruning as it relates to flow control and as it is implemented in multicast protocols.

### 3. GROUP UTILITY AND PRUNING

#### 3.1 Discussion of Group Utility Function

Let there be one sender<sup>3</sup> and  $n$  receivers in the multicast group, and let  $M = \{1, 2, \dots, n\}$  denote the receiver set. From the sender to

<sup>2</sup> We came across the papers [9], [11], and [10] after we started our studies, through reviewers of our work.

<sup>3</sup> The multi-sender problem can be considerably more complicated, depending on how the multiple senders share the multicast distribution tree. For simplicity, we assume the N

each receiver  $k$ , let  $R_k$  denote the best-effort available bandwidth for the multicast flow to reach that receiver. We refer to  $R_k$  as the *receive rate* for receiver  $k$ . Without pruning, the multicast flow control algorithm should result in adopting a group data transmission rate  $R_M$  such that

$$R_M = \min_{k \in M} R_k$$

As a result of applying pruning, the flow control may potentially operate at a higher data transmission rate  $R_S$  where  $S$  denotes a subset of  $M$ , and consists of receivers that are left after pruning.

When the group data rate is  $R$ , let  $U_k(R)$  denote the utility function for receiver  $k$ , and let  $U_S(R)$  denote the utility function for the subgroup  $S$  (remaining after pruning). The pruning decision is the selection of a subset  $S$  so as to maximize the group utility:

$$\max_{S \subseteq M} \left( U_S(R_S) + \sum_{k \in M} U_k(R_S) \right)$$

The subgroup utility function captures the importance of having a particular subgroup remain in the multicast session. Consider the analogy with scheduling a meeting. Often the criterion is given as follows: it must include individuals A, B and C; it is desirable to have D and E attend; and F, G and H should be invited but their attendance is not important. If the meeting is carried out using multicast, then the above criterion can be captured using a suitable set of subgroup utility functions. This aspect of receivers' utility is not covered by the Inter-Receiver Fairness function proposed by [9], [11], and [10].

Let us consider a numerical example. Suppose the utility functions are defined as:

$$\begin{aligned} U_S(R) &= 0 \\ U_k(R) &= 0 \quad \text{if } R_k < R \\ U_k(R) &= \frac{1}{R_k - R + 1} \quad \text{if } R_k \geq R \end{aligned}$$

This means that there is no special utility for any subgroup  $S$  in not being pruned; each receiver's utility is such that if the sending rate exceeds its receiving rate, then the utility is zero, but otherwise the utility increases as the sending rate approaches the receiver's rate.

Given this utility function, if  $M = \{1, 2, 3, 4\}$  and the receiver rates  $R_k, k=(1,2,3,4)$  are  $\{4, 3, 2, 1\}$ , then the best thing to do is to not prune any receivers and operate at group rate  $R_M=1$ . On the other hand, if the receiver rates are  $\{2, 2, 2, 1\}$ , then the best thing to do is to prune the fourth member and operate at group rate  $R_{\{1,2,3\}}=2$ .

sender problem is the same as N single sender problems and do not address the N sender problem in this paper.

Basing pruning decisions on optimizing the group utility is called *optimal pruning*. In our view, this is mostly a theoretical perspective of the problem. The work in [9], [11] and [10] proposes a utility function called *inter-receiver fairness* for the purpose of grouping receivers into multiple subgroups and delivering different services to them. In the extreme case, there would be two subgroups – one with full service and the other with no service, which corresponds to our pruning problem. This is an admirable attempt at the complex problem of optimal grouping and pruning. The suitability of the proposed metric, we feel is subject to debate. It is a weighted sum of each receiver’s utility, thus allowing particular receivers to be considered more important; it does not, however, capture the utility of any particular subgroup of receivers, as allowed by the general utility function above. The inter-receiver fairness function also requires measuring (or estimating) the group size and each receiver’s receive rate. The accurate and timely measurement of these parameters on a continuing basis could be difficult.

### 3.2 A Practical and TCP-friendly Criterion for Pruning

For practical purposes, a much simpler form of pruning criterion suffices in many situations. A pruning data rate,  $R_{prune}$ , is configured. Any receiver  $k$  whose rate  $R_k$  is less than  $R_{prune}$  is subject to pruning. This criterion is equivalent to some reasonable utility functions, for example:

$$\begin{aligned}
 U_k(R) &= 0 \quad \text{if } R < R_{prune} \\
 U_k(R) &= 0 \quad \text{if } R > R_k \\
 U_k(R) &= 1 \quad \text{if } R_{prune} \leq R \leq R_k \\
 U_S(R) &= 0 \quad \text{if } R \neq R_S \\
 U_S(R) &= 1 \quad \text{if } R = R_S
 \end{aligned}$$

In other words, there is no utility to a receiver when the rate of transmission is either below the pruning rate, or higher than the rate that a receiver is capable of receiving. Furthermore, there is no marginal utility in addition to the constant (=1) utility of being able to keep up with the transmission. The subgroup utility function makes sure that the (unpruned) subgroup  $S$  receives at the subgroup rate,  $R_S$ , rather than a rate between  $R_S$  and  $R_{prune}$ .

This pruning criterion affords us some important simplifications to the problem. Specifically, it is not necessary to measure each receiver’s receive rate  $R_k$ , which, due to its dynamic nature may be hard to measure accurately. Instead, it is only necessary to know which receiver’s rate is less than the pruning rate  $R_{prune}$  at the time pruning is deemed necessary. This latter condition, while still tricky to determine, can usually be established based on various flow control information already available.

Another important consideration is TCP-friendliness. By establishing a pruning rate and letting the slowest receiver above the pruning rate dictate the group rate and discarding the rest of the receivers whose rates are below the pruning rate, the multicast group attempts to co-exist with TCP traffic fairly. See [1] for a more detailed discussion of fairness between unicast and multicast

flows. Note, if a data rate based on a general function of receive rates is used, it may not be TCP-friendly. This is simply because the general function of multicast group members’ receive rates does not take into consideration of the rates of other TCP flows. In this sense, the IRF as discussed in [11] is not TCP-friendly in this strict sense. In [10], the IRF is expanded to include TCP flows sharing the network with the multicast flow; but their solution is not practical, as it requires knowledge of the rates for the TCP flows. By simply adopting the receive rate of the slowest receiver (above the pruning rate), we have separated the pruning decision from the flow control decision, and left to the flow control algorithm to ensure TCP-friendliness.

In the rest of the paper, we will discuss the practical aspects of implementing this simple pruning criterion in a reliable multicast transport protocol.

## 4. PRUNING ALGORITHMS

### 4.1 Pruning and Its Relationships to Multicast Transport Protocols

In section 3, we discussed pruning in abstract terms, and proposed pruning receivers that cannot receive at a session-defined pruning data rate. The implementation of this criterion can be very straightforward or moderately complicated depending on the goal of the underlying transport protocol of which pruning is a part. Several cases are listed below:

- 1) In one extreme, if the transport is semi-reliable and the session does not have any sense of its membership (e.g. [5]), then pruning can be totally the responsibility of the individual receivers. Each receiver monitors its ability to receive at the prevailing session data rate. Upon discovering that the data rate is too high, the receiver leaves the multicast group voluntarily. This voluntary departure implements pruning.
- 2) Alternatively, the transport protocol may be assisted by routers based on the Generic Router-Assist framework as in [3], [17]. In this case, the routers can help accurately identify which links are too slow and prune all the receivers behind those links.
- 3) In another class of protocols, the session keeps track of its receiver membership and provides reliability service and flow control based on the response of all its members. Some examples of this protocol are RMTP [13] and TRAM [2]. In this case, each receiver has a *monitor* that keeps some state about the receiver. When a receiver is pruned (or leaves a group), its monitor must be aware of the event. This departure relieves the session from certain reliability services and the responsibility of the flow control to react to this receiver. There is more to pruning in this case; minimally, it involves both the receiver to be pruned as well as its monitor.
- 4) Finally, the addition of repair traffic can complicate the identification of slow receivers to be pruned. A slow receiver that cannot keep up with the minimum session data rate is likely to cause packet losses and retransmissions. Unfortunately the retransmission traffic can cause more packet losses. Depending on the topology and strategy for doing repairs (particularly in those protocols in (3)), the

retransmission traffic may affect some receivers more than others, creating complicated scenarios for identifying the receivers that truly deserve getting pruned.

In this paper, we study mostly case (3). We discuss and show some initial results for case (4) in Section 6. For other protocols (such as (1) and (2)), although the measurement techniques may be different, the general concepts we develop still apply.

For protocols in (3), the receivers are often organized into a (repair) tree for scalability. We first describe some basic operating assumptions about the repair tree and how it is used for pruning. Each interior node of the tree is a repair head that provides retransmission when requested. There are regular (or periodic) communications between the receivers and the repair heads. The receivers send one ACK (acknowledgement) packet for every  $W$  data packets received.<sup>4</sup> The repair heads, in turn, send ACKs to their parents on the same regular basis. These ACK packets provide a feedback channel from the receivers to their repair heads and to the sender. For example, some information about the minimum receive rate can be passed up the tree using the ACK packets, with the repair heads aggregating that information along the way.

Each repair head maintains a buffer (also called cache) of data packets potentially needing retransmission. When all children of the repair head have acknowledged up to a particular sequence number, all packets older than and including that sequence number can be removed from the cache. We refer to this as *advancing the cache*. If the acknowledgement window is  $W$  (packets), under normal operating conditions the repair head needs to be able to buffer at least  $W$  packets, typically more.

In order to support potential late-joiners, the repair head can optionally delay removing those packets that have been acknowledged by all its children, as long as the cache occupancy has not reached a threshold that is dangerously close to the maximum limit. This threshold is normally several times of  $W$ .

Pruning is performed normally by a repair head sending a message to a child receiver to *disown* that receiver. A receiver may also resign; in some protocols where the state is kept at the receiver, the receiver resignation constitutes pruning. In our study below, no receiver resigns. A third scenario we do consider is when a receiver departs ungracefully (crashes). In this situation, its repair head relies on a timer, and a probe (hello) message to ascertain the receiver is no longer responding, and therefore removes the crashed receiver from its list of children.

The multicast sender includes dynamic session information in the header of a regular DATA packet to help repair heads decide when to prune. Specifically, the header can contain a pruning flag that indicates the sender suggests pruning.<sup>5</sup>

In our studies, we considered basically two pruning algorithms, which differ in whether primarily local or global information is used to decide when and whom to prune:

---

<sup>4</sup> This is the normal case. When there are many losses, or when a long time elapses between successive data packets, the receivers may send additional ACKs.

<sup>5</sup> The pruning flag can also be set in any other control packet that is multicast to all the receivers in the session.

A. *Local pruning algorithm*: Each repair head makes pruning decisions based on locally available information and conditions. This includes:

- cache occupancy information, and which member caused the oldest packet in the cache;
- sender current rate (which could be passed down in the DATA packet header by the sender);
- loss rate for each member;
- Whether a receiver has sent an ACK in the last timeout period and/or responded to the probe message.

B. *Global pruning algorithm*: Information about receiver rates (absolute or relative) is collected at each receiver. This information is then aggregated by the repair heads of the tree and propagated towards the sender. The sender determines when it is time to prune, and propagates a pruning signal and some information that helps identify the slowest receiver(s) to help the repair heads determine which receivers to prune. The sender does not necessarily know about all receivers, but only some characteristic that corresponds to the slowest receiver. Based on this characteristic, each repair head can then determine if one of its children should be pruned.

The development of the global pruning algorithms is mostly motivated by the reasons described in (4); in other words, to help more accurately identify slow receivers in the face of repair traffic.

We describe these algorithms (and their variations) in more detail in the subsequent sections.

Finally, it should be clarified again that the sending rate is dynamically controlled by a flow and congestion control mechanism built into the transport protocol. Many such control schemes have been proposed or discussed elsewhere (such as [6] and [7]), and it is not the main subject of this paper. The pruning algorithm does not set the sending rate; the pruning algorithm only comes into effect when the sending rate (established by flow control) is consistently under the pruning rate.

## 4.2 The Local Pruning Algorithm

In the local pruning algorithm, the pruning decision is decentralized. The pruning criteria are based on locally available state information. We describe several pruning criteria below. These following pruning criteria are complementary, and can be put together to form the local pruning algorithm.

### 4.2.1 Pruning based on cache occupancy

The first pruning criterion is indicated when the cache occupancy reaches a threshold and yet no packet can be removed from the cache (because some children have not acknowledged the packets in the cache). The child receivers that are holding up the cache advance are pruned. This is a minimalist pruning criterion, in part because there is no calculation of receive rates or any other state information, and in part because the repair head is forced to start pruning (else there is no place to put the newly arriving DATA packets).

Although there is no guarantee that this local pruning condition coincides with the stated pruning criterion that the pruned member cannot keep up with the pruning rate, it is most likely that is the case. Keeping a receiver that ought to be pruned (based on

its low receive rate) will eventually cause the repair head's cache to fill up anyway.<sup>6</sup>

Intuitively, we can see that pruning based on the cache state ought to work, is efficient and simple, but probably suffers from being too conservative and hence slow in performance.

Note, however, when used in conjunction with a window-based flow control algorithm which sometimes allows the congestion window to determine the effective sending rate, this cache state based pruning criterion may not work. In a window-based flow control algorithm (for example, see [6] or [12]), the congestion window prevents the flow's losses from growing beyond a level controlled by the maximum window size. Besides, the slow receiver would cause the congestion window to shrink and, as a result, reduce the effective sender rate to be below the pruning rate without pruning occurring. In this scenario, the repair head's cache threshold has to be small in comparison to the congestion window size in order for pruning to occur.

The effectiveness of this pruning criterion depends on the setting the threshold to a level commensurate with the acknowledgement and congestion window. Each repair head compares the *highest consecutively received* (HCR) sequence number for all its members. The threshold is applied against the distance between two HCR values for two different members.

#### 4.2.2 Pruning based on locally observed loss rate

Since the goal is to prune any receiver whose receive rate is below the minimum data rate, it is natural to try to measure the rate for each receiver, and use that to determine if a receiver should be pruned. Based on the sender's multicast transmissions, however, the receivers are basically all measuring the sender's sending rate.

One metric that does distinguish the slow receivers from the fast ones is the packet loss rate. The fast receivers tend to have zero or very low loss rate, while the slow receivers will on average have higher loss rates the slower the receiver is.

In [11], there is a discussion of how receive rates (referred to as isolated rates) are measured. It can be summarized by the following formula:

$$r_k = \begin{cases} r_s * (1 - p_k) & \text{if loss} \\ r_s / (1 - p_T) & \text{if no loss} \end{cases}$$

Here  $r_k$  is the estimated rate for the receiver  $k$ ;  $r_s$  is the sender's rate;  $p_k$  is the loss rate for receiver  $k$ , and  $p_T$  is a constant loss tolerance for the whole group. When there is no loss at a receiver, it is not possible to measure that receiver's rate. What is known is that that receiver's rate is no less than the current sending rate,  $r_s$ . A constant factor,  $1/(1 - p_T)$  is applied to the sending rate as an incremental way to best approximate the receive rates (with no loss). As a result, the Inter-Receiver Fairness value derived using this receive rate measurement technique is a very coarse approximation.

In our case, we are only interested in finding (and ranking) receivers who are slower than the minimum data rate. That list of

receivers is the list of those that still suffer from packet losses when the sender is sending at the minimum data rate. The receive-rate ranking of these receivers is simply the inverse of their ranking according to their loss rates. While this method of identifying and ranking slow receivers is quite simple, it is only statistical since there is some randomness in the correlation of packet losses to the deviation of a receiver's rate from the minimum data rate. For this reason, we continuously compute a receiver loss rate over many windows of packets, and apply exponential smoothing to the computation. Either the receiver or its repair head can perform this computation since they both have the required information.

Using this pruning criterion, the repair head prunes those receivers when their loss rate exceeds a threshold and at the same time the sender is sending at the minimum data rate. This pruning criterion directly meets the requirement of removing those receivers that cannot receive at the minimum data rate.

#### 4.2.3 Pruning based on timeouts

In addition to the above pruning criteria, the repair head needs to monitor and detect the situation when a member leaves without notice (for example the program crashed for some reason). This is a special case of pruning – the receiver is *pruned (by virtue of departing)* first and then the repair head must learn this reality as quickly as possible so that the departed receiver does not impact the performance of the rest of the multicast group.

As soon as a receiver missed sending a regular ACK message, its repair head sets a timer and starts probing the receiver for an ACK using repair group management messages. If the receiver fails to respond within a number ( $m$ ) of probes, the receiver is considered pruned. This process takes

$$m * RTT_i$$

where  $RTT_i$  is the round-trip time from the repair head to the departed receiver. This round-trip time can be measured also using the same repair group management messages. Due to the potential large variation in round-trip times, the measurement is exponentially smoothed and often a configured lower bound is applied.

### 4.3 The Global Pruning Algorithm

In a multicast setting, it is not clear what the *ripple effect* of a slow receiver is on other receivers. For example, the additional repair traffic (often via multicast) and control traffic induced by the slow receiver may cause packet losses for other receivers (under possibly different repair heads) which would not happen otherwise. When these packet losses occur at the minimum data rate, the repair head using the local pruning algorithm may prune these unfortunate receivers prematurely.

One way to avoid pruning receivers unnecessarily due to the ripple effect of the worst receiver is to prune the worst receiver(s) sequentially. In other words, try to prune the worst receiver(s) in each step and then determine if there are still receivers satisfying the pruning condition; if so, repeat the process until the pruning condition goes away. Let us call this the Worst Receiver First (WRF) algorithm.

<sup>6</sup> This assumes the sender continues to send at the pruning rate.

To determine the worst receiver(s) requires some global coordination. For example, let the pruning criterion be based on packet losses at the time when the sender is sending at the minimum data rate. Suppose there are two repair heads. The receiver loss rates for the 1<sup>st</sup> repair head are {1%, 4%, 5%}, whereas the receiver loss rates for the 2<sup>nd</sup> repair head are {1%, 1%, 2%}. Without global information, the 2<sup>nd</sup> repair head cannot tell if the receiver that is losing 2% packets is locally the worst or globally the worst. The WRF algorithm can discard the worst and some other receivers sufficiently close to the worst in each step. In this example, the first repair head may consider the receiver with 4% loss and 5% loss sufficiently close and prune both of them in the same step.

Given a repair tree, the global pruning algorithm can be easily implemented. The repair tree is typically used to gather other multicast group-wide information, such as data packet stability, congestion feedback, and group membership information. The computation of the group-wide slowest receive rate can be piggybacked on the existing feedback information flow.

The metric to rank receivers can again be based on cache occupancy or loss rate measurement, as in the local case. In our implementation used for experimentation, we use the loss rate to determine the slowest receiver globally. Each receiver computes its loss rate for each data acknowledgement period, and applies exponential averaging – this computation is the same as that in the local pruning algorithm. This information is passed to the repair head, piggybacked on ACK packets. The repair heads aggregate this information (compute the loss rate of the slowest receiver in its subtree) and pass it up towards the sender. The sender eventually gets the loss rate of the slowest receiver.

More explicitly, the ACK packet carries the following two pieces of information:

- Aggregated (exponentially averaged) loss rate of the slowest receiver in the subtree below (inclusive of) a repair head.
- Subtree indicator - an indication that the slowest receiver is some receiver in the subtree rather than the local receiver.

In the reverse direction, the DATA packet (or any other packet that is multicast to the whole group) can be used to distribute global pruning information. This includes:

- Exponentially averaged loss rate of the slowest receiver in the whole tree.
- Prune indication – indicating that it is highly probable that the slowest receivers in the tree are not capable of keeping up with the pruning rate, and requesting repair heads to prune children whose loss rate closely matches the value for the whole tree.

A typical implementation of a global pruning algorithm is shown in pseudo-code in Figure 1, in terms of what the receivers, the repair heads, and the sender do.

Note that pruning is triggered when there is congestion, and the sender notices that the average data rate is lower than the pruning rate. The exact implementation of `calculateAve(history)` is an interesting engineering detail that can affect the aggressiveness of pruning. In our implementation, we used a simple moving window algorithm. Both the global algorithm and the local algorithm are triggered the same way.

```
Receivers:
//
//When it is time to ACK:
//
curLossRate = numLost / numReceived;
// w is a smoothing factor between 0 and 1
aveLossRate = w * curLossRate +
              (1-w) * aveLossRate;
subtreeIndicator = false;
SendAck(aveLossRate, subtreeIndicator);
```

```
Repair heads:
//
//Upon receiving an ACK from its child:
//
UpdateChildLossRate();
ComputeWorstLossRateOfChildren();

//
//When it is time to ACK:
//
CurLossRate = numLost / numReceived;
// w is a smoothing factor between 0 and 1
aveLossRate = w * curLossRate +
              (1-w) * aveLossRate;
aggregatedLoss = worst(aveLossRate,
                      LossRateOfChildren);
if (aggregatedLossRate == aveLossRate)
    subtreeIndicator = false;
else
    subtreeIndicator = true;
SendAck(aggregatedLoss, subtreeIndicator);

//
//When DATA packet contains prune indicator:
//
children = matchLossRate(slowestLossRate);
if (children != null)
    Prune(children);
```

```
Sender:
//
//At sending of each DATA packet
//
totalSent = totalSent + size(packet);

//
//At every ackWindow boundary
//
curTime = now();
keepTrack(curTime, totalSent, history);
aveRate = calculateAve(history);

//
//Upon receiving a congestion report:
//
if ( aveRate < pruneRate )
    Send_prune_signal();
```

**Figure 1. Sample pseudo-code fore the global pruning algorithm**

## 5. TEST RESULTS

### 5.1 Evaluation Criteria

The following factors are important considerations when evaluating the effectiveness of various pruning algorithms:

- Time to discover and prune slow (or crashed) receiver(s) must be minimized. Pruning is closely tied to flow control. When a slow (whose rate is below the prune rate) or crashed receiver is not pruned from a multicast session, the flow control algorithm is forced to operate at the minimum data rate (prune rate); the longer this condition persists the more it affects the overall multicast performance.
- Probability a receiver is pruned by mistake must be minimized. The determination of when to prune and whom to prune are often stochastic. This is especially true when a receiver's rate is barely above the pruning rate. There is usually a trade-off between this factor and the time-to-prune factor, as more time usually allows a more accurate determination.
- Efficiency (with respect to processing and bandwidth overheads) and simplicity.

### 5.2 Test Configurations

The algorithms described in the last section were implemented as part of a reliable multicast transport protocol (and its flow control). In this section, we describe the test environment for our experiments. We used ten different machines, one as sender and the rest as receivers, connected by a high-speed campus network,<sup>7</sup> so that the network is never the bottleneck in any of our tests. The sender and all the receivers can easily sustain a throughput of 200KB/sec. So when the maximum data rate is set to 200KB/sec, it is roughly the achieved average throughput. To emulate a slow receiver, we have implemented a packet filter sitting in between the UDP socket and our RM transport. This packet filter continuously measures the data receive rate as packets are received. Whenever the measured receive rate goes above a pre-configured threshold, the packet filter randomly drops packets. We call this packet filter the receive rate emulator. All our experiments involve configuring some receivers to have a slow receive rate and monitoring the pruning behavior.

In order to study the effectiveness of both the local and global pruning algorithms, we manually configured the receivers into a repair tree as shown in Figure 2.

### 5.3 Basic Behavior of the Pruning Algorithm

For calibration purposes, we first ran multicast tests from the sender to one receiver at a time. In each case, the maximum data rate was set to 200KB/sec, and 5 MB of data was transmitted. All the tests produced 200KB/sec throughput since the sender, receiver and network can all sustain that rate of transmission.

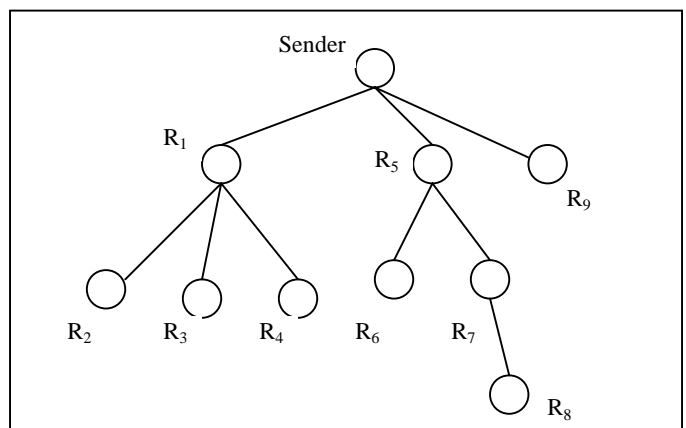
Then we ran a series of tests to calibrate the receive rate emulator. In these tests, the maximum data rate was set to 200KB/sec, the minimum data rate to 1KB/sec, and the emulator's

receive rate was varied from 40KB/sec up to 70KB/sec. The achieved average throughput<sup>8</sup> was as below, all units in KB/sec.

**Table 1. Receive rate emulation calibration**

| Emulator rate | Average Throughput | Range        |
|---------------|--------------------|--------------|
| 40            | 38.9               | [37.4, 39.9] |
| 50            | 46.0               | [45.4, 47.1] |
| 60            | 56.2               | [52.6, 61.5] |
| 70            | 62.1               | [59.3, 64.7] |

As we can see, the emulator is not perfect. Given a configured receive rate to emulate, the achieved throughput is always somewhat below the specified emulator rate. This is not surprising, since some time was spent by the flow control algorithm to ramp up, and some time was spent in retransmissions (shown as range) is also significant. This is partly due to the short test duration used for calibration. For the purposes of our experimentation, this level of consistency and accuracy is adequate.



**Figure 2. Repair tree**

Next, we did a simple test of the global pruning algorithm, using the repair tree shown in Figure 2, with receiver R<sub>3</sub> configured to receive at 60KB/sec. The maximum data rate is set at 200KB/sec, and the pruning (minimum) data rate is set to 70KB/sec. The sender sends 5MB of data, pauses for 10 seconds, and repeats this cycle. As expected, R<sub>3</sub> was pruned soon after it joins the group. After each prune, R<sub>3</sub> was programmed to pause for 30 seconds and rejoin the multicast group. The presence of R<sub>3</sub> temporarily slowed down the group until R<sub>3</sub> was pruned again. Figure 3 was captured from our monitoring program that tracks the instantaneous sending rate (dark line) and number of packets allowed to send by the congestion window (gray line) observed at the sender.<sup>9</sup>

<sup>7</sup> The tests were mostly done during weekends and evenings. While there may still be some other traffic on the network, the results are largely reproducible.

<sup>8</sup> Each was based 3 to 5 samples.

<sup>9</sup> Each sample is taken 0.5 seconds apart.

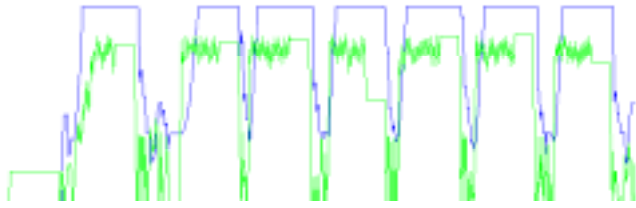


Figure 3. Basic pruning behavior

When the slow receiver is absent, the sending rate reaches the maximum 200KB/sec and the window opens wide. Whenever the slow receiver joins the group, the sending rate drops to around 50-60KB/sec and the window closes down significantly. The exact workings of the congestion control algorithm that control the sending rate and congestion window will be reported in a future paper. This illustration is included to show that the pruning algorithm is doing its job.

## 5.4 Local and Global Pruning

In this section, we describe experimental results of testing both the local and global pruning algorithms, using the multicast group organized in the same repair tree as shown in Figure 2. To make the example interesting, we configure three of the receivers to have lower emulated rates as below:

Table 2. Three slow receivers' rates

|                |          |
|----------------|----------|
| R <sub>2</sub> | 50KB/sec |
| R <sub>3</sub> | 60KB/sec |
| R <sub>6</sub> | 70KB/sec |

Sender sends 25MB of data with a maximum data rate of 200KB/sec. Only these three slow receivers can limit the sender's rate from ramping up to 200KB/sec. The pruning rate is varied from 40KB/sec up to 70KB/sec.

### 5.4.1 Local Pruning

In these experiments, local pruning was done by using the *highest consecutively received* (HCR) sequence number as the differentiator between receivers. In a separate simulation study, we found the use of HCR to determine the slowest receiver is more effective than using exponentially smoothed loss rate as the metric. (Of course, the use of HCR is the same as cache occupancy, but a more exact definition). When a pruning signal is received from the sender (piggybacked in a data message), a repair head first determines,  $h$ , the highest value of HCR out of all its children (including the repair head itself). A repair head then proceeds to prune all members with an HCR value that satisfies

$$HCR < h - \text{threshold}$$

The measurement results are summarized in Table 3. All rates are in units of KB/sec, and all times are in units of seconds. The value of *threshold* was one acknowledgement window (32).

Table 3. Pruning behavior using local algorithm

| Prune rate | Ave thput | Time to prune R <sub>2</sub> | Time to prune R <sub>3</sub> | Time to prune R <sub>6</sub> | Time to prune R <sub>9</sub> |
|------------|-----------|------------------------------|------------------------------|------------------------------|------------------------------|
| 40         | 50.1      |                              |                              |                              |                              |
| 50         | 47.3      |                              |                              |                              |                              |
| 60         | 60.6      | 5                            | 231                          |                              |                              |
| 70         | 175.4     | 10                           | 18                           | 39                           | 39                           |

When the prune rate is 40KB/sec, even the most limiting receiver (R<sub>2</sub>) can sustain that rate. The algorithm behaved correctly by not pruning any receiver and achieved a throughput of 50.1KB/sec, roughly corresponding to R<sub>2</sub>'s rate. When the prune rate is 50KB/sec, the local algorithm was on the *lenient* side and again did not prune R<sub>2</sub>. This arguably is still the correct behavior; the achieved throughput was a sluggish 47.3KB/sec. When the prune rate is 60KB/sec, both R<sub>2</sub> and R<sub>3</sub> were pruned. Finally, when the prune rate is 70KB/sec, the local algorithm managed to prune all three slow receivers very quickly. Unfortunately, it also erroneously pruned R<sub>7</sub>. The sender was then able to transmit at the maximum data rate 39 seconds after the session started, and a throughput of 175.4KB/sec was achieved.

The pruning behavior was basically correct for pruning rate up to 60KB/sec. When multiple slow receivers exist, they are pruned in the right order. For the 50KB/sec case, the algorithm acted on the conservative side. Although the 50KB/sec case finished with slightly lower throughput than the 40KB/sec case, it can be attributed to the imprecise receive rate emulation.

But for the 70KB/sec case, the algorithm acted on the overly aggressive side and pruned one extra member (R<sub>9</sub>), at the same time when R<sub>6</sub> was pruned.

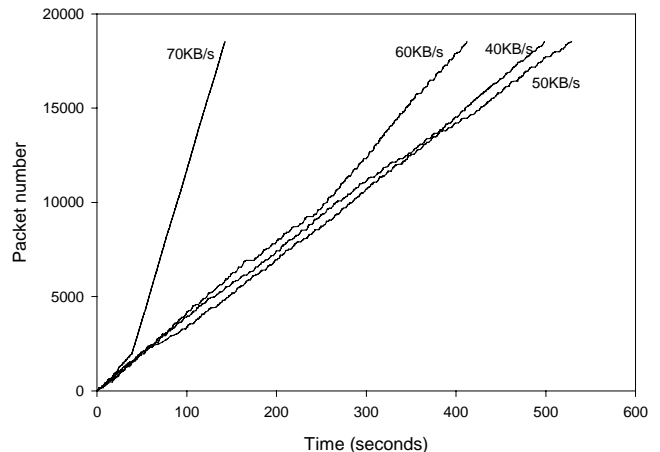


Figure 4. Local pruning example

Figure 4 plots the sender's packet sequence number versus time. The plots for pruning rates equal to 60KB/sec and 70KB/sec are notable. In the former case, the curve changed slope after 231 seconds into the experiments; and in the latter case, there is a

much more pronounced change of slope after 39 seconds into the experiments, when all slow receivers were pruned.

To further investigate the possible reason for incorrect pruning, we repeated the test with pruning rate equal to 70KB/sec three more times. Two out of the four tests had an incident of wrong pruning and the other two were perfect.

We believe setting the threshold to 1 acknowledgement window is too aggressive. This is particularly the case since we implement a *staggered* acknowledgement strategy so that not all receivers acknowledge at the same point in an acknowledgement window. This makes the receivers that acknowledge at a later point in the acknowledgement window (relative to the point the pruning condition first started) more vulnerable.

To correct this situation, we tried additional experiments with the threshold set to 1.5 and 2 acknowledgement windows. The results are as shown in tables 4 and 5.

**Table 4. Local pruning with threshold set to 1.5 acknowledgement windows**

| Prune rate | Average throughput | Time to prune R <sub>2</sub> | Time to prune R <sub>3</sub> | Time to prune R <sub>6</sub> |
|------------|--------------------|------------------------------|------------------------------|------------------------------|
| 40         | 48.7               |                              |                              |                              |
| 50         | 49.8               |                              |                              |                              |
| 60         | 54.0               | 20                           |                              |                              |
| 70         | 159.6              | 23                           | 13                           | 48                           |

When the threshold is set to 1.5 acknowledgement windows, no receivers are pruned by mistake. The pruning is a bit conservative for lower pruning rates. For the pruning rate equal to 70KB/sec case, the order of pruning the slow receivers was not in accordance to the receive rates, but the slow receivers were all pruned correctly.

When we increased the threshold to 2 acknowledgement windows, however, the local pruning algorithm became overly conservative. This is as expected. In fact, as we discussed earlier, when the threshold is set to some arbitrary level depending on available cache size, local pruning may not happen at all.

**Table 5. Local pruning with threshold set to 2 acknowledgement windows**

| Prune rate | Average throughput | Time to prune R <sub>2</sub> | Time to prune R <sub>3</sub> | Time to prune R <sub>6</sub> |
|------------|--------------------|------------------------------|------------------------------|------------------------------|
| 40         | 49.0               |                              |                              |                              |
| 50         | 48.4               |                              |                              |                              |
| 60         | 47.6               |                              |                              |                              |
| 70         | 54.5               | 232                          | 55                           |                              |

These experiments demonstrate that while the local pruning algorithm is simple and fast, it needs careful tuning and calibration for correct operations. The appropriate threshold value to use also seems to depend on how far apart the pruning rate and slow receive rates is.

### 5.4.2 Global Pruning

We then tested the global pruning algorithm using the same network scenario as above, with the same statically configured

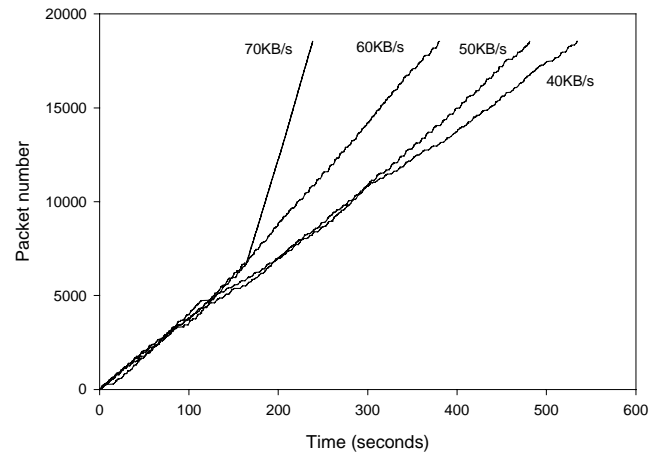
repair tree as in Figure 2 and slow receiver configured as in Table 2. The corresponding results are shown in Table 6.

**Table 6. Pruning behavior using global pruning algorithm**

| Prune rate | Average throughput | Time to prune R <sub>2</sub> | Time to prune R <sub>3</sub> | Time to prune R <sub>6</sub> |
|------------|--------------------|------------------------------|------------------------------|------------------------------|
| 40         | 46.8               |                              |                              |                              |
| 50         | 51.9               | 245                          |                              |                              |
| 60         | 65.7               | 3                            | 109                          |                              |
| 70         | 104.8              | 25                           | 112                          | 164                          |

The pruning behavior was perfect. The time to prune was somewhat longer compared to those achieved by the local pruning algorithm. All in all, the results here are more consistent and the global algorithm seems to be more robust.

Figure 5 shows the corresponding dynamic behavior of the global algorithm. In this case, the curves for 50KB/sec, 60KB/sec and 70KB/sec all exhibited a slope change after the pruning was done.



**Figure 5. Global pruning example**

## 5.5 Some Other Observations

One of the key observations from our experiments is that a basic trade-off in the pruning strategy is in its timeliness and tolerance to temporary slow-downs (accuracy). In the case of the local algorithm, we go into some detail in discussing how a key parameter, the threshold, affects this trade-off.

Some other parameters can also affect this trade-off, for example, how the average sending rate is calculated. We use a sliding window (of 5 seconds) to compute the average. The sliding window size determines how sensitive the average is to the instantaneous rate versus the historic average.

We also did a lot of experimentation with getting the probe-based local pruning right. This mechanism has to be there for all pruning algorithms since it is used to discover crashed receivers or receivers that for some reason stops responding. This timeout (based on RTT measurements) mechanism is very important to get right, as it can affect the whole multicast group's progress.

In our experiments, we used a packet filter to emulate slow receivers. This emulator drops packets randomly, whereas in the real world a slow receiver is more likely to drop packets back-to-back when buffer overflows. This may affect the “accuracy” of our results.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, we studied an important aspect of multicast flow control that has not received much attention in the literature. The problem is how to allow a multicast flow to adopt a group-wide optimal rate rather than the rate of the slowest receiver. This can be accomplished by pruning (or discarding) the slow receivers that cannot keep up with the group-wide optimal rate. In our study, the group optimal rate is the pruning rate that is configured.

We investigated different algorithms, implemented some, and reported test results using a test network with ten receivers forming a three level repair tree. From these experiments, we found that both local and global pruning algorithms can work adequately well. The local algorithm requires more careful calibration and tuning, but under perfect conditions it seems to outperform the global algorithm slightly.

We also discussed that a fundamental trade-off in the pruning strategy is between timeliness and accuracy of pruning. We discussed how to adjust the algorithms for this trade-off.

There are still many issues worthy of further research.

The difference between the local and global algorithms can be further investigated using more different network scenarios. Simulation may provide a way to experiment with more diverse and complicated setting.

As we briefly discussed in the beginning of section 4, one of the issues is the effect of the repair traffic on the correct determination of the slowest receivers. This potentially impacts the local algorithm more, since the global algorithm follows the Worst Receiver First algorithm.

It would be interesting to study other factors that might make a particular receiver more susceptible to being pruned, such as distance from the sender, whether being a repair head and its associated workload would increase the likelihood of being pruned, and so on. These factors can then be factored into the design of the pruning algorithm and the algorithms for building a good repair tree.

## 7. REFERENCES

- [1] D.Chiu, “Some Observations on Fairness of Bandwidth Sharing”, Sun Microsystems Laboratories Technical Report, TR-99-80, <http://www.sun.com/research/techrep/1999>, to appear in ISCC2000.
- [2] D. Chiu et al, “TRAM: A Tree-based Reliable Multicast Protocol,” Sun Microsystems Laboratories Technical Report, TR-98-66, <http://www.sun.com/research/techrep/1998>.
- [3] B.Cain, T. Speakman, D.Towsley, “Generic Router Assist (GRA) Building Block – Motivation and Architecture,” IETF Internet Draft draft-ietf-rmt-gra-arch-00.txt. Work in progress.
- [4] C. Diot, W. Wabbous, and J. Crowcroft, “Multipoint Communication: A Survey of Protocols, Functions and Mechanisms,” IEEE Journal of Selected Areas in Communications, Vol 15, No. 3, April 97.
- [5] S. Floyd et al, “A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing”, ACM Transactions on Networking, November 1996.
- [6] J. Golestani “Fundamental Observations on Multicast Congestion Control in the Internet,” Bell Labs, Lucent Technology, paper distributed to RM Research Group in August 1998.
- [7] M. Handley, S.Floyd, and B.Whetten, “Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC)”, paper distributed to RM Research Group in 1999, work in progress.
- [8] S. Herzog, S. Shenker, and D. Estrin, “Sharing the ‘Cost’ of Multicast Trees: an Axiomatic Analysis,” Transactions on Networking, 5, pp 847-860. 1997.
- [9] T. Jiang, M. Ammar, and E.Zegura, “Inter-Receiver Fairness: A Novel Performance Measure for Multicast ABR Sessions”, in Proceedings of SIGMETRICS’98, 1998.
- [10] T.Jiang, M.Ammar, and E.Zegura, “On the Use of Destination Set Grouping to Improve Inter-receiver Fairness for Multicast ABR Sessions”, in Proceedings of INFOCOMM’00, 2000.
- [11] T. Jiang, E. Zegura, and M. Ammar, “Inter-Receiver Fair Multicast Communication Over the Internet”, in Proceedings of NOSSDAV’99, June 1999.
- [12] M. Kadansky et al, “Tree-based Reliable Multicast (TRAM),” IETF Internet Draft draft-kadansky-tram-02.txt, Jan 2000. Work in progress.
- [13] J.Lin, S.Paul, “RMTP: A Reliable Multicast Transport Protocol,” IEEE INFOCOMM’95, 1995.
- [14] H.Moulin and S.Shenker, “Strategyproof Sharing of Submodular Costs: Budget Balance versus Efficiency”, preprint, 1997. At <http://www.aciri.org/shenker/cost.ps>.
- [15] Jose F de Resende et al, “Fully Reliable Multicast in Heterogeneous Environments”, Proceedings of PfHSN’96, Sophia Antipolis, France, Oct 1996.
- [16] D. Rubenstein, J. Kurose, D. Towsley, “The Impact of Multicast Layering on Network Fairness,” ACM SIGCOMM’99, Cambridge, MA, Sept 1999.
- [17] T. Speakman et al, “PGM Reliable Transport Protocol” IETF Internet Draft draft-speakman-pgm-spec-03.txt. Work in progress.
- [18] T.Turletti and J.C. Bolot, “Issues with Multicast Video Distribution in Heterogeneous Packet Networks,” in Proceedings of the 6<sup>th</sup> Int. Workshop on Packet Video, Portland, OR, Sept 26-27, 1994, pp. 301-304.
- [19] L.Vicisano, L. Rizzo, J. Crowcroft, “TCP-like Congestion Control for Layered Multicast Data Transfer,” IEEE INFOCOMM’98, San Francisco, CA, Mar 28-Apr 1, 1998.
- [20] D. Waitzman, C. Partridge, and S. Deering, “Distance Vector Multicast Routing Protocol,” RFC 1075, IETF, Nov 1988.