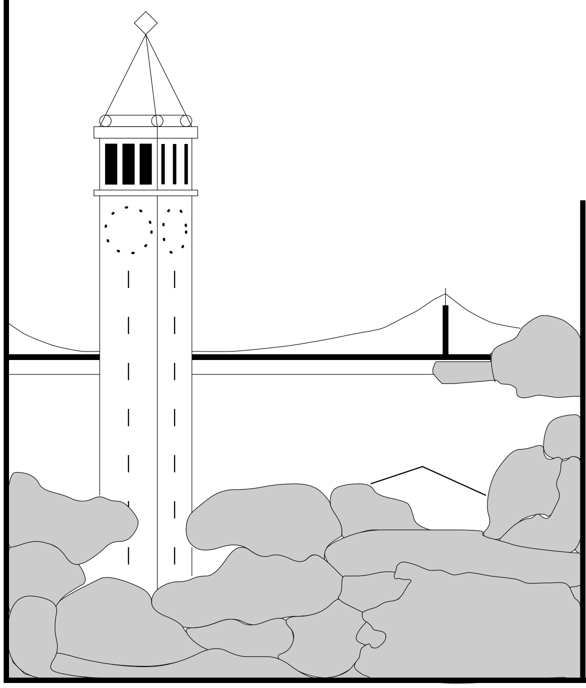
Search Party:
Using Randomcast for Reliable Multicast with Local Recovery

Adam M. Costello *Steven McCanne*



Report No. UCB//CSD-98-1011

August 1998

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Search Party: Using Randomcast for Reliable Multicast with Local Recovery

Adam M. Costello
http://www.cs.berkeley.edu/~amc/

Steven McCanne
http://www.cs.berkeley.edu/~mccanne/

26 Aug 1998

Abstract

IP multicast is an efficient means of sending to a group, but the packets are sent unreliably. Some applications, like distributed whiteboard and news articles, require detection and retransmission of lost packets. In order to scale to large groups, *local recovery* is necessary to avoid involving the entire group in the repair process for packet losses affecting small regions of the distribution tree. While many current research efforts have attempted to devise local recovery schemes that rely only on the existing service model, we believe that extending the multicast forwarding service could enable viable and highly scalable local recovery mechanisms. To investigate this open issue, we propose a new randomized forwarding service called *randomcast*, and build upon it a loss recovery protocol called *Search Party*. Starting with the local recovery structure of the very scalable LMS scheme [PPV98], we use randomized forwarding to greatly improve robustness at a modest cost in overhead and/or retransmission delay (the trade-off between the two costs is fine-tunable). Analysis predicts that as the group size N increases, overhead will increase by at most $\log N$ and retransmission delay will be unaffected. Simulation experiments show that both increase very little as N grows from 8 to 64, and confirm the tunability of the trade-off.

1 Introduction

The IP multicast service [DC90] allows a source to send packets unreliably to a group of N members much more efficiently than sending N unicast packets. Some applications, like real-time audio and video broadcasting, are delay-sensitive and loss-tolerant, and thus well-suited to IP multicast service. But other applications that could benefit from efficient multicasting require reliable delivery. Some are real-time, like distributed whiteboard [FJL+97] and multiplayer games, while others are bulk transfers, like software updates and news articles. To support these applications, we need a reliable multicast protocol that performs *loss recovery*—the detection and retransmission of lost packets. While a reliable multicast protocol should also perform *congestion avoidance* [MRBP98], this paper restricts its attention to loss recovery. A good summary of reliable multicast work is included in [LLG96].

1.1 Model

In order to discuss multicast forwarding, we will refer to the following model of the internet. An internet topology is a bipartite graph in which the nodes are *routers* and *networks*, and the edges are *interfaces*. Each interface connects a router to a network. Packets originate and terminate at *endpoints*. In reality an endpoint lies within a router (usually a degenerate router called an "end host"), but we will consider the endpoint to be adjacent to the router. Multicast packets are forwarded from a *source* to group *members* along a *distribution tree*. For each source/group pair, there is a tree (a subgraph of the internet topology) in which the source is the root[1] and the members are the leaves (figure 1). When a packet arrives at a node, a copy of the packet is

[1] Some proposed multicast routing protocols like PIM [DEF+94], CBT [BFC93], and BGMP [KRT+98] can use *shared trees*, in which the source is not the root, but this paper focuses on source-rooted trees. See section 7.2 for more discussion of shared trees.

This work is supported by the National Science Foundation (ANI-9734515, CDA 94-01156), the Advanced Research Projects Agency (N66001-96-C-8508) and California MICRO.

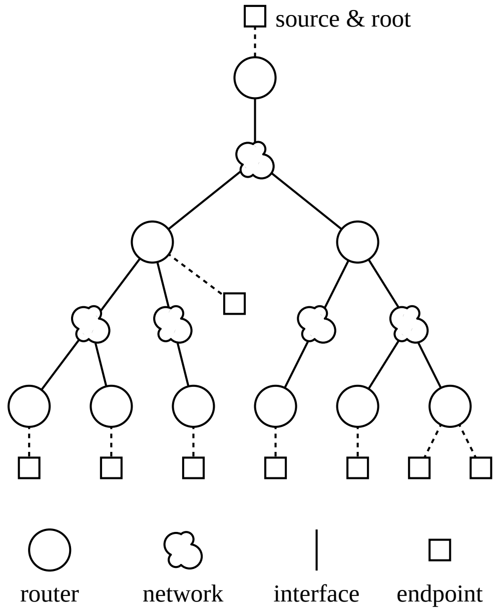


Figure 1: Example multicast distribution tree.

forwarded to each child. Whenever a packet is lost, it fails to reach the members in the *loss subtree*—the subtree below the point where the loss occurred.

1.2 Implosion

If a loss recovery scheme is to scale to arbitrarily large groups, its first task is to avoid the well-known *implosion* effect. Reliable protocols designed for a single sender and receiver are often based on automatic repeat request (ARQ), where the receiver sends feedback to the sender, so the sender knows when it needs to send retransmissions. But if we extend ARQ to multicast, and multiple receivers send feedback to a single sender, the resulting implosion of packets can overwhelm the sender or the network links near it.

Implosion has been successfully addressed in two pioneering works on reliable multicast. Scalable Reliable Multicast (SRM) [FJL+97] uses *multicast damping*. Members multicast repair requests to the entire group, but use random timers to ensure that only about one request is sent at a time. Other members waiting to send the same request see the first one and suppress their own. Any member capable of responding to a request may do so, and random timers are used again to limit the number of responses to about one.

The Reliable Multicast Transport Protocol (RMTP) [LP96] avoids implosion via *ACK fusion*. The members are organized into a hierarchy. ACKs are sent not to the source, but to the parent member, who merges the ACKs before sending them to its own parent.

1.3 Local Recovery

After implosion avoidance, another essential task is *local recovery*. If a lost packet affecting a relatively small subtree triggers a recovery effort involving the sender, or involving too much of the tree, the protocol cannot scale to arbitrarily large trees.

SRM implementations do not yet provide local recovery, because every recovery involves the entire tree. Section 7.2 of [FJL+97] sketches a few potential ideas for local recovery in SRM, but their efficacy is not evaluated.

In RMTP, internal nodes in the hierarchy (called Designated Receivers) send retransmissions from their data caches. RMTP therefore provides local recovery *if* the hierarchy of members is closely correlated to the multicast distribution tree. However, information about the topology is not available above the routing layer, so it is not clear how the hierarchy can be constructed, nor how it can track routing changes.

A mechanism proposed in RMTP to aid in local recovery is *subtree multicast*, which defines a new IP packet type to let any host tunnel a multicast packet to a router, allowing the host to send a packet to a subtree rooted at any router in a multicast tree.

1.4 LMS

Lightweight Multicast Services (LMS) [PPV98] provides implosion avoidance and local recovery like RMTP, but unlike RMTP it uses the multicast tree itself rather than a separate hierarchy, and thereby circumvents the need to construct and maintain the hierarchy. To make this possible LMS extends the routers with a new but simple forwarding service. Each router in a multicast tree selects one of its child links to be its *replier link*. (LMS is described in terms of point-to-point links rather than networks and interfaces.) A packet received from a non-replier link is forwarded to the replier link, while a packet received from the replier link is forwarded to the

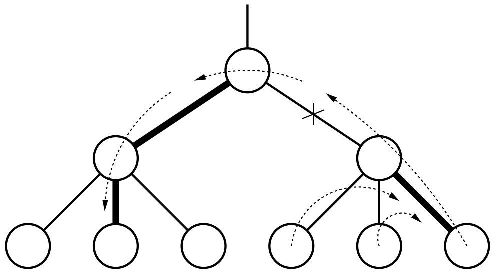


Figure 2: The paths taken by three request packets after a packet loss in LMS. The replier links are indicated by heavy lines.

parent. When a member detects a lost packet, it uses this forwarding service to send a request for a retransmission. The remarkable consequence of the forwarding rule is that when every member in a loss subtree issues a request for the missing packet, exactly one request will escape the subtree, even though no one (not even the routers) knows where the loss occurred or how many requests were sent. The other requests go to members inside the loss subtree, who are unable to respond (figure 2).

The router that bounces an upward-moving request back downward is called the *turning point* (notice that a request packet can reverse direction at most once). The turning point router inserts into the request its own address and an identifier for the link on which the request arrived. This allows the recipient of the escaping request to respond with a *directed multicast* packet, which is very similar to a subtree multicast packet, except that it also includes the link ID, which is used by the turning point router to further restrict the audience. Whereas the subtree multicast service forwards a packet to a subtree rooted at a router, directed multicast service forwards a packet to a subtree rooted at a link. When the loss occurs on a non-replier link, the response goes only to the loss subtree; when the loss occurs on a replier link, the turning point is higher than necessary and the response goes to a larger subtree.

Even though the LMS design introduces new network functionality, the extensions in no way violate clean layering principles—the routers do not see any transport-layer information like sequence numbers, nor do the endpoints know anything about topology.

1.5 Robustness and Randomization

Although LMS does not construct an explicit hierarchy like RMTP, there is still an implicit hierarchy: every subtree has just one member who sends requests on its behalf and one member who responds to them. Therefore both schemes suffer from concentration of responsibility, because a single member bears the entire burden of sending retransmissions for a particular lossy link, and one malfunctioning member can adversely affect a large number of other members below it in the hierarchy. Also, in LMS the locality of retransmissions depends heavily on where losses occur and on which links are chosen to be replier links.

Robustness can be improved through randomization. If requests are directed to other members chosen randomly rather than deterministically, responsibility is diffused, and it becomes impossible for one malfunctioning member to have a significant impact on a large number of other members. Whereas LMS uses a simple deterministic forwarding service, we propose a simple randomized forwarding service called *randomcast*, and show one approach that exploits randomcast to build a scalable, efficient, and robust multicast loss recovery protocol called *Search Party*.

1.6 Search Party Overview

Search Party follows the structure of LMS: requests for retransmissions are sent up the tree toward the root and bounced back down with inserted information about the turning point. Unlike LMS, Search Party requests are forwarded randomly with randomcast. As in LMS, responses are sent to the subtree below the interface through which the request arrived at the turning point node. Randomizing the routing of requests trades off performance (a constant factor) for increased robustness. Because responsibility is diffused evenly throughout the group, a malfunctioning member has a limited impact on its near neighbors, and a vanishing effect on the entire group. Also, the burden of sending retransmissions for a particular lossy link is shared among many members, rather than a single replier or designated receiver. The performance penalty takes the form of increased retransmission delay and/or increased recovery overhead, and applications with varying delay sensitivities can fine-tune the trade-off between the two.

When a member detects a loss, it continually sends requests until a retransmission arrives. In effect, each member conducts a random search for the missing data. Because no one knows where the loss occurred, no one knows how broad a search ought to be conducted; however, because all and only those members affected by the loss participate in the search, they automatically form a *search party* of just the right size and scope.

In the remainder of this paper, we present the details of Search Party and randomcast. Sections 2 and 3 describe randomcast and subcast forwarding, and section 4 explains the loss recovery protocol built upon them. Results of analysis and simulation are presented in sections 5 and 6. Plans for future work appear in section 7, and conclusions in section 8.

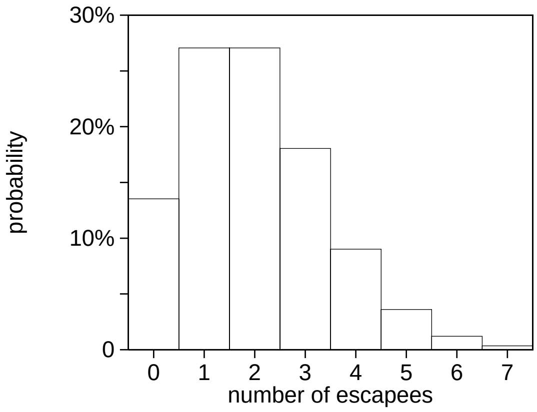


Figure 3: Probability that k randomcast packets escape a large subtree in which each leaf sends two packets (Poisson distribution with mean 2).

2 Randomcast Forwarding

Randomcast is a proposed new service that forwards packets randomly inside a multicast distribution tree. A regular multicast packet contains a multicast group address and a source address, which routers use to infer which distribution tree to use, but the mapping from source/group to tree is private to the routers (it may be a source-rooted tree or a shared tree). A randomcast packet specifies the tree explicitly by including a root address. When a randomcast packet arrives at a node, it may be forwarded[2] to any neighbor in the tree except the one the packet came from. Whenever a node acts as a turning point (receives a randomcast packet from a child and forwards it to another child), it inserts into the packet information sufficient to address the subtree below the arrival interface (see section 3).

The probability distribution used to select the outgoing interface is critical to the behavior of systems using the randomcast service. We evaluated two distributions: uniform, and weighted by subtree population. We found that the weighted distribution is better for choosing between the parent and children, while uniform distribution is better for choosing among children.

[2] This is not a problem if the node is a router, but is slightly tricky if the node is a network, because networks are generally not intelligent. The best solution is to have routers adjacent to the network perform the forwarding decisions on the network's behalf. Alternatively, networks can be ignored, and routers can be considered neighbors of each other.

2.1 Parent or Child?

When a randomcast packet arrives at a node X from a child C, the node first decides whether to forward the packet to the parent or to another child. If the distribution is uniform, the probability that the parent is selected is $1/c$, where c is the number of children of X. (The probability is not $1/(c+1)$ because the node is forbidden from returning the packet to the child it came from.) If the distribution is weighted by subtree population, the probability that the parent is selected is $L(C)/L(X)$, where $L()$ denotes the number of leaves descended from a node (1 if the node is itself a leaf). An essential feature of both distributions is that the sum of the probabilities over all the children is 1.

Some multicast routing protocols already require routers to be aware of their children in the tree, so each router could occasionally communicate with its children and obtain a count of the number of leaves below each one, computing the sum for its own use and its parent's. The EXPRESS multicast service proposal [HC] suggests that routers keep track of subtree populations. The uniform distribution does not require knowledge of subtree populations, but the weighted distribution has two attractive properties.

First, the probability that a randomcast packet originating at any leaf below node X reaches X's parent is $1/L(X)$, regardless of the topology. This diffuses responsibility for sending requests evenly among the members, and also eases analysis: If m randomcast

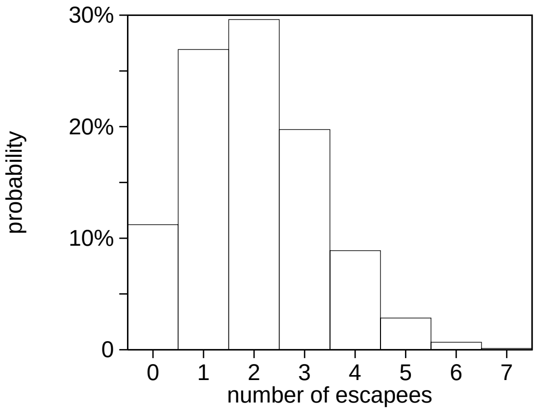


Figure 4: Probability that k randomcast packets escape a subtree of six leaves in which each leaf sends two packets (binomial distribution with $n = 12$ and $p = 1/6$). Compare this to figure 3.

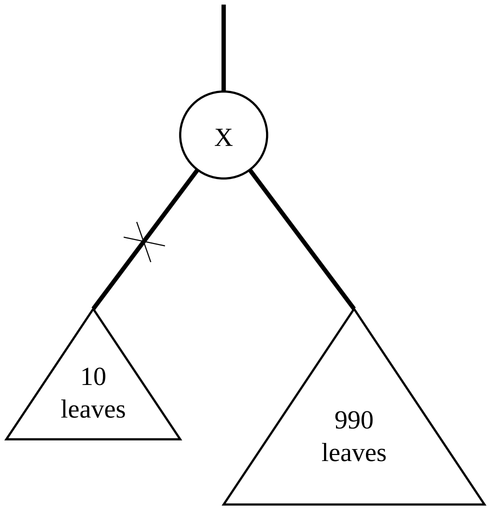


Figure 5: A packet loss in an imbalanced tree.

packets are sent from each of the L leaves in a subtree, the number of packets that escape the subtree has a binomial distribution with parameters $n = mL$ and $p = 1/L$, for which the expected value is $np = m$ [Str89]. It is well-known that as n increases while np remains constant, the the binomial distribution assymptotically approaches the Poisson distribution with $\lambda = np$. Therefore, as L increases, the probability that k packets escape approaches $\frac{m^k}{k!e^m}$ (which is independent of L). The Poisson distribution is fairly narrow, as shown in figure 3, meaning that the number of escapees is fairly predictable. The binomial distribution is slightly narrower for smaller subtrees, but even with only six leaves it has about the same shape, as shown in figure 4.

The second attractive property of the weighted distribution is locality. Consider the scenario of figure 5. With a uniform distribution, there is a 50% chance that a request from the smaller subtree will be forwarded to X's parent, causing the response to be delivered to the members in both subtrees, of which 99% are not interested. With a weighted distribution, there would be only a 1% chance of this undesirable occurrence. If the loss occurs just above the larger subtree, the weighted distribution will almost always forward the request to X's parent, but the response will still be of interest to 99% of the recipients.

Thus, for the choice between parent and children, Search Party uses the weighted distribution.

2.2 Which Child?

When a packet is to be forwarded by a node X to one of its children, either because it arrived from the parent or because the node has already decided against forwarding to the parent, there is again a choice between uniform and weighted distributions. With a uniform distribution, the probability of selecting child C is $1/c$ or $1/(c-1)$ depending on whether the packet arrived from a child (and is therefore forbidden from being forwarded to that child). With a weighted distribution, the probability is $L(C)/L(X)$ if the packet arrived from the parent. If the packet arrived from child A, the probability that it gets forwarded to child C is $L(C)/[L(X) - L(A)]$. It follows that before the node has decided whether to forward to the parent, if it is using a weighted distribution for both choices, the probability that it forwards to C is $L(C)/L(X)$.

A node using the weighted distribution for selecting children will tend to distribute randomcast packets evenly among its descendent leaves, but packets from many incoming edges can get concentrated onto one outgoing edge. If that happens several times in a succession of nodes, the downward traffic on a single edge could grow exponentially. With the uniform distribution, a node distributes packets evenly among its downward edges, avoiding that hazard. Because trees with more leaves tend to be deeper, the weighted distribution will tend to route packets to farther away leaves, whereas the uniform distribution will tend to choose shorter paths, leading to smaller round-trip times. For these reasons, Search Party uses the uniform distribution for selecting children.

3 Subcast Forwarding

Subtree multicast, proposed for use with RMTP, and directed multicast, used by LMS, are similar services. We use the generic term *subcast* to refer to any forwarding service that allows packets to be sent to a subtree. Recall from section 1.1 that the nodes of an internet topology alternate between routers and networks, so that either may be the root of a subtree. Ideally, subcast would support both router-rooted and network-rooted subtrees; if not, a loss just above a non-addressable node X will have to be treated as c losses, one above each of X's child nodes.

The root node of the destination subtree is called the *exploder*. There are at least two ways the sender of a subcast packet could specify the desired exploder: by address, or by distance. If addresses are used, a member can subcast a packet to a subtree that it does not belong to, as required by LMS and Search Party. If distances are used, the exploder must be on the path from the member to the root, so a member can subcast only to subtrees that it belongs to. In this case, there is a simple workaround: a response can be sent indirectly via unicast to the requestor, who then relays the response via subcast to the desired subtree. Therefore, Search Party does not constrain the design of the subcast service.

4 Loss Recovery Protocol

4.1 Basic Operation

Search Party is an error recovery protocol built on top of the randomcast and subcast forwarding services. Each group member is responsible for detecting missing packets and for obtaining retransmissions by sending requests until a response arrives.

A member is able to detect losses because the source multicasts *updates* that contain sufficient state information, like the greatest sequence number sent so far. The updates can be piggybacked with data except during lulls in the data stream.

Because the internet can reorder packets, each member must estimate the delay variance, which it can do if the source puts a timestamp in every update[3]. Each member chooses as a *loss detection threshhold* a multiple of the estimated standard deviation of the delay. After observing a gap in the data, the member waits for the threshhold to elapse before concluding that a packet has been lost. The choice of multiplier is a trade-off between delay and probability of false detection. If the network layer were to provide almost-always-in-order delivery, a threshhold of zero would be sensible.

After a member detects a loss, it acts as a *requestor*, emitting randomcast *request packets* as a Poisson process with mean rate[4] μ until it gets a corresponding *response packet*. Like the loss detection threshhold, μ is a trade-off between overhead and delay. If requests are sent more often, the expected time to escape the loss subtree is lower, but more requests are likely to escape unnecessarily and generate duplicate responses. Each member may choose its own value of μ. During a recovery, the effective μ value is the average over all members in the loss subtree.

Any member who receives a request acts as a *responder*. If the responder has the requested data, it subcasts a response packet to the subtree indicated in the request, otherwise it does nothing. When the source receives a request, it behaves the same way a member would, except that it uses multicast rather than subcast (the request has no turning point).

The following sections provide more details of the protocol, pertaining to response time estimation, refined requestor and responder behavior, and message stability.

4.2 Response Time Estimation

If a request generates a response, the amount of time between the emission of the request and the reception of the response by the requestor is called the *response time* (r). Each member must estimate the expected response time (both the mean and variance) for recoveries involving that member, for reasons to

[3] We need not assume that the source clock and the member clocks all run at the same rate, because each member can estimate its drift relative to the source as in NTP [Mil95], and every time value appearing in a packet (except the timestamps in the updates) can be converted to source clock ticks by the sender and then to local clock ticks by the receiver.

[4] Packet emission rates are measured in inverse time units. For example, 1/s means one packet emission per second.

be given. The best-guess estimate is called $\hat{r}$, and the estimate of the upper bound (at some confidence level) is called r_{hi}. Suppose a requestor puts a timestamp in an outgoing request, and the timestamp is copied into the response by the responder. This allows the requestor, upon receiving the response, to calculate the response time[5]. It does not, however, allow the other recipients of the response to calculate the response time. We could have the requestor announce the response time via a subcast packet, but we can avoid that extra traffic by altering the representation of the timestamp.

Let every update contain a unique ID, and every request contain a *time reference*, which is a pair of numbers: the ID of the most recently received update, and an offset, which is the time difference between the reception of that update and the sending of the request. Every response contains a time reference copied from the request. The requestor can calculate the response time r from the time t_{resp} that the response is received, the time t_{ref} that the referenced update was received (if it was received), and the offset:

$$r = t_{\mathrm{resp}} - (t_{\mathrm{ref}} + \mathit{offset})$$

Furthermore, every recipient of the response can perform the same calculation and get approximately the same result, because the difference between the times two members receive a packet is usually about the same for any multicast or subcast packet, so the variations in t_{resp} and t_{ref} will cancel out.

4.3 Null Requests

If loss rates are low, there may be inadequate losses to generate accurate response time estimates. To allow adaptation to continue in the face of low loss rates, members may send null requests as a Poisson process with mean rate ρ. A null request always generates a null response. If every member chooses the same ρ, every member can expect to receive null responses at a rate between ρ and $\rho \cdot (1 + \ln N)$, where N is the number of members. The responses will have turning points distributed evenly over the balanced heights of the tree, so there will be samples representing the full range of possible response times. The *balanced height* of a node X is $\ln L(X)$; for any balanced tree, the balanced height is proportional to the actual height. Like μ values, ρ values effectively get averaged: a member's choice of ρ affects itself most, and other members decreasingly with distance.

[5] It is not sufficient for the requestor to remember when the request was sent, rather than insert a timestamp, because it sends multiple requests that must be disambiguated [KP91].

4.4 Request Burst

One use of the response time estimates is to refine the request behavior. According to the earlier description, a requestor sends requests as a Poisson process until a response arrives. Therefore, requests scheduled to be sent after the response arrives will not be sent, but requests scheduled to be sent before the response arrives are destined to be sent, so they might as well have been sent as soon as the loss was detected, to reduce the recovery time. Of course, the requestor does not know ahead of time when the response will arrive, but it can take a reasonably conservative guess that it will be at least $\hat{r}$ after the loss is detected. The requestor expects to send $\hat{r}\mu$ requests during the first $\hat{r}$, so it should send them a burst when the loss is detected, then wait $\hat{r}$ before beginning the Poisson process. Note that $\hat{r}\mu$ is generally not an integer; the requestor issues $\mathrm{trunc}(\hat{r}\mu)$ requests, plus one more with probability $\mathrm{frac}(\hat{r}\mu)$. The request burst can decrease retransmission delay and duplicate responses if the loss detections happen at about the same time, and the actual response times are all about the same. If this is not the case, the burst will not significantly alter the aggregate behavior of the members, compared to the original Poisson-only strategy.

4.5 Internal Response Suppression

There is a problem with duplicate responses coming from *inside* the loss subtree. Before any response arrives, requests are circulating inside the loss subtree, arriving at members who, being inside the loss subtree and lacking the data, ignore the requests. But when the first response arrives, there are requests are already in flight, about to arrive at members who just now received the same retransmission that has already satisfied the requestors. We do not wish for these requests to generate responses.

To prevent these internal duplicate responses, each member remembers the last time it received a retransmission of the data, and ignores requests that predate the retransmission. If a responder has the requested data, and has seen a retransmission of the requested data, it maps the request's time reference to a moment $t_{\mathrm{req}} = t_{\mathrm{ref}} + \mathit{offset}$, and compares t_{req} to the arrival time t_{rtx} of the most recent retransmission of the requested data[6]. If it were the requestor performing the comparison, then obviously the request has already been satisfied iff $t_{\mathrm{req}} \leq t_{\mathrm{rtx}}$. It is actually the responder who performs the comparison, but because the difference between the times two members receive a packet is usually about the same for any multicast or subcast packet, the variations in t_{rtx} and t_{ref} will cancel out. To allow for some noise, the responder compares $t_{\mathrm{req}} - t_{\mathrm{rtx}}$ against a positive threshhold rather than zero. The threshhold is a multiple of the combination of the requestor's and responder's estimated standard deviations of data packet delay, so the requestor must include its estimate in the request packet. Because variances tend to be additive, standard deviations are combined as follows:

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$$

4.6 Message Stability

When may the source and members delete old data and associated state information? If the source wishes to insure that all members receive the data, there must be a group membership protocol and a scheme for merging ACKs, problems not addressed by Search Party. If it is left to each member to insure that it has received all the data, a simple course-grain timeout may suffice. The source discards old data when the deletion timeout expires, and includes information in the updates so that members can discard their old data also. If a member cannot recover some lost data before it is discarded, it might mean that the information is too old to be useful anyway, or that something is so seriously broken that an entirely different avenue for obtaining the information should be tried instead, like fetching from an archive.

[6] If the responder is unable to calculate t_{req} because it has not received the referenced update, it must use a default guess. For example, it could use *now* if low delay is valued more than low overhead, or $now - r_{\mathrm{hi}}$ if low overhead is valued more, or perhaps $now - \hat{r}/2$.

5 Performance Analysis

An application using Search Party must choose the rate at which it sends requests. The choice can be a function of the response time estimates, but what function? Before we can answer that question, we need to understand the effects that these variables (μ, $\hat{r}$, r_{hi}) have on the performance of the system after a loss. Sections 5.1 through 5.4 give analytic results for four effects: response duplication, request traffic, response locality, and retransmission delay. Section 5.5 then shows how the equations can be used to tune μ.

In order to make the analysis tractable, we make the following simplifying assumptions:

- Every member in the loss subtree detects the loss at the same time, called time 0.
- Every member in the loss subtree has the same value for $\hat{r}$.
- The response time r is the same for every *escapee* (request that escapes) from the loss subtree.
- A response arrives simultaneously at all members.
- Requests and responses are not lost.

These assumptions are approximations, so the results they imply are approximations. The results are stated here without proof; the proofs and derivations appear in appendix A.

5.1 Response Duplication

After a loss is detected, members in the loss subtree send requests until the first response arrives. The members then stop sending requests, but some of the requests already in flight can escape the loss subtree and generate duplicate responses. Let μ denote the average request rate among the members in the loss subtree. If $r \leq \hat{r}$, the expected number of escapees (and hence, responses) is

$$x(r) = \hat{r}\mu + (1 + r\mu)e^{-\hat{r}\mu}$$

If $r \geq \hat{r}$, the expected number of escapees is

$$x(r) = r\mu + (1 + \hat{r}\mu)e^{-\hat{r}\mu}$$

Our best guess is $r = \hat{r}$, at which point the two expressions are equal, yielding

$$x = \hat{r}\mu + (1 + \hat{r}\mu)e^{-\hat{r}\mu}$$

A pessimistic prediction is $r = r_{\mathrm{hi}} > \hat{r}$, yielding

$$x = r_{\mathrm{hi}}\mu + (1 + \hat{r}\mu)e^{-\hat{r}\mu}$$

If the network is not working at time 0 (perhaps because the loss was caused by a persistent malfunction), all of the immediate requests will be lost, rendering the request burst ineffective. The expected number of escapees is then

$$x(r) = 1 + r\mu$$

which is worse (greater) if $r \geq \hat{r}$, though not necessarily if $r < \hat{r}$. Therefore, an extra-pessimistic prediction is

$$x = 1 + r_{\mathrm{hi}}\mu$$

This prediction is an especially safe upper bound on x, because it does not rely on the network working at time 0, nor on the accuracy of $\hat{r}$, and assumes that the response time of every escapee is r_{hi}, though in actuality almost all will be less than r_{hi}.

5.2 Request Traffic

During a recovery, members inside the loss subtree receive requests that they ignore. Let L be the number of leaves in the loss subtree. Since each request has a $1/L$ chance of escaping, and the expected number of escapees is x, the expected number of requests generated must be xL. The $x \cdot (L-1)$ requests that do not escape are redistributed among the L leaves. Therefore, on average, a member in the loss subtree expects to receive about the same number (x) of requests as responses.

During the recovery each edge in the loss subtree carries upward request traffic with mean rate equal to the average μ value among the members below itself. Above the loss point, the request traffic dissipates—this is the sense in which the search for the missing data automatically has the proper scope, even though no one knows where the loss occurred. If each member uses the same μ value, the downward request traffic on each edge in the loss subtree must be less than 2μ. Although the μ values are not necessarily the same, this result shows that the randomcast routing itself does not concentrate request traffic. As a corollary, the null request traffic on every edge is ρ upward and less than 2ρ downward.

5.3 Response Locality

[PPV98] defines *exposure* as the ratio of the number of members who receive a response over the number of members in the loss subtree. Exposure is thus a measure of non-locality, and perfect locality corresponds to an exposure of 1. For Search Party, the expected exposure of a loss depends on the topology, but cannot exceed $1 + \ln \frac{N}{L}$.

For comparison with LMS, consider a balanced tree with losses equally likely on all edges. In this scenario, LMS gives upward-moving requests a $1/c$ chance of continuing upward. Search Party also gives a $1/c$ chance (regardless of how losses are distributed), so the expected exposure is the same $(1 + \frac{c-1}{c} \log_c \frac{N}{L})$.

Response duplication (see above) and non-locality are orthogonal phenomena that each cause more response packets to be delivered than necessary. The total expansion factor is the product of the individual factors.

5.4 Retransmission Delay

The probability that no retransmission has arrived before time t is

$$p(t) = \begin{cases} 1 & \text{for } t < r \\ e^{-\hat{r}\mu} & \text{for } r \leq t \leq r + \hat{r} \\ e^{(r-t)\mu} & \text{for } t \geq r + \hat{r} \end{cases}$$

The middle section of the distribution relies on the effectiveness of the request burst, which is questionable. Therefore, when choosing a μ value, applications should consider the probability distribution only for $t \geq r_{\mathrm{hi}} + \hat{r}$. The best-guess prediction is thus

$$p(t) = e^{(\hat{r}-t)\mu}$$

and the pessimistic prediction is

$$p(t) = e^{(r_{\mathrm{hi}}-t)\mu}$$

The average retransmission delay (expected value of t when the retransmission arrives) is

$$y(r) = r + (\hat{r} + \tfrac{1}{\mu})e^{-\hat{r}\mu}$$

which is $\frac{x}{\mu}$ if $r \geq \hat{r}$, which works for our best-guess ($r = \hat{r}$) and pessimistic ($r = r_{\mathrm{hi}}$) predictions. Of course, the delay cannot be predicted at all if the network is not working at time 0.

Notice that the delay is independent of N. To consider the asymptotic behavior as r increases, we can ignore the distinction between r, $\hat{r}$, and r_{hi}. Since x is a function of $r\mu$, holding x constant makes $r\mu$ constant, so that the expected retransmission delay is $O(r)$, whereas it was simply r in LMS.

5.5 Examples of Tuning

To see how application programs can tune μ to suit their own needs, we consider three hypothetical applications.

News articles Suppose this application can tolerate long delays, but wants a probability of less than one in a billion that a retransmission will take longer than ten minutes (perhaps because the source will discard the data by then). Round-trip times of more than ten seconds are unheard-of in the internet, so for simplicity the application could let $r_{\text{hi}} = 10\,\text{s}$ and not bother to estimate it. The request burst (sending $\hat{r}\mu$ requests immediately and then waiting $\hat{r}$) has a negligible benefit when μ is small, as it will turn out to be. Therefore, the application could forgo the burst by letting $\hat{r} = 0$ and not bothering to estimate it. The pessimistic delay requirement is

$$\begin{aligned} e^{(r_{\text{hi}} - 600\,\text{s})\mu} &\leq 10^{-9} \\ (10\,\text{s} - 600\,\text{s})\mu &\leq \ln 10^{-9} \\ \mu &\geq \tfrac{1}{28.5\,\text{s}} \end{aligned}$$

The expected number of responses per loss is, extra-pessimistically, $x = 1 + r_{\text{hi}}\mu = 1.35$, but if r has a more realistic value, say $1\,\text{s}$, then $x = 1.04$. The expected retransmission delay is, pessimistically, $y = r_{\text{hi}} + \frac{1}{\mu} = 38.5\,\text{s}$.

Distributed whiteboard Suppose this application wants retransmissions to take less than $t_{\max} = 2\,\text{s}$ from the time the loss is detected, with a failure probability of $p = 10\%$. The pessimistic delay requirement is

$$\begin{aligned} e^{(r_{\text{hi}} - t_{\max})\mu} &\leq p \\ \mu &\geq \frac{\ln p}{r_{\text{hi}} - t_{\max}} \end{aligned}$$

The application must adjust μ in response to changes in r_{hi}. Suppose the estimates $\hat{r}$ and r_{hi} are $0.2\,\text{s}$ and $0.8\,\text{s}$ respectively. Then $\mu = 1.92/\text{s} = \frac{1}{521\,\text{ms}}$. The expected number of responses per loss is $x = 1 + r_{\text{hi}}\mu = 2.54$ (extra-pessimistic) or $x = \hat{r}\mu + (1 + \hat{r}\mu)e^{-\hat{r}\mu} = 1.33$ (best-guess). The expected retransmission delay is $y(r) = r + (\hat{r} + \frac{1}{\mu})e^{-\hat{r}\mu} = r + 0.49\,\text{s} = 1.29\,\text{s}$ (pessimistic) or $0.69\,\text{s}$ (best-guess). If network performance were to improve, causing r_{hi} to decrease, this application would respond by decreasing μ to decrease the overhead while maintaining the delay constraint.

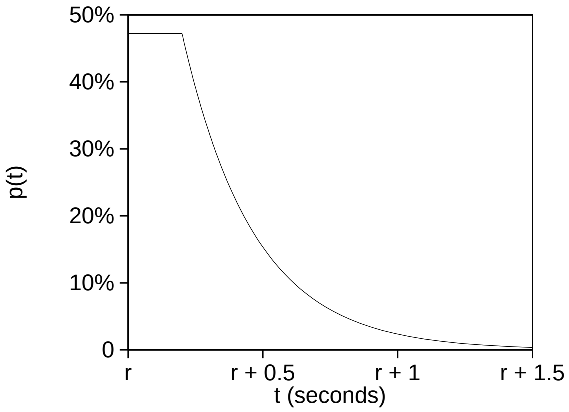


Figure 6: The probability $p(t)$ that no retransmission has yet arrived at time t for the multiplayer game application. Notice that the left edge of the graph is time r, not time 0.

Multiplayer game The internet is already too slow for this application. It wants to spend at most $x_{\max} = 4$ responses per loss, on average, to get retransmissions as quickly as possible. The extra-pessimistic response requirement is

$$\begin{aligned} x = 1 + r_{\text{hi}}\mu &\leq x_{\max} \\ \mu &\leq \frac{x_{\max} - 1}{r_{\text{hi}}} \end{aligned}$$

As with the whiteboard application, the game application must adjust μ in response to changes in r_{hi}. If we again suppose that $\hat{r}$ and r_{hi} are $0.2\,\text{s}$ and $0.8\,\text{s}$ respectively, then $\mu = 3.75/\text{s} = \frac{1}{267\,\text{ms}}$. The expected number of responses is 4.00 (extra-pessimistic) or 1.58 (best-guess). The expected retransmission delay is $r + 0.22\,\text{s} = 1.02\,\text{s}$ (pessimistic) or $0.42\,\text{s}$ (best-guess). The probability that the response has not arrived before time t is shown in figure 6 in terms of r. If network performance were to improve, causing r_{hi} to decrease, this application would respond by increasing μ to decrease the delay while maintaining the overhead constraint.

Unified tuning algorithm The methods used by the whiteboard and game applications for calculating μ can be combined into a single algorithm. Let the application choose a target retransmission delay $t_{\max}$, an acceptable failure probability $p > 0$, and a maximum average number of responses per loss, $x_{\max} > 1$. The response constraint overrides the delay constraint. The protocol library can then calculate

$$\mu = \begin{cases} \frac{x_{\max}-1}{r_{\mathrm{hi}}} & \text{if } r_{\mathrm{hi}} \geq t_{\max} \\ \min\left(\frac{\ln p}{r_{\mathrm{hi}}-t_{\max}}, \frac{x_{\max}-1}{r_{\mathrm{hi}}}\right) & \text{otherwise} \end{cases}$$

If the application is required to choose p and $x_{\max}$ such that

$$p \leq e^{1-x_{\max}}$$

then the library will always be obeying the admonition to use the $p(t)$ distribution only for $t \geq r_{\mathrm{hi}} + \hat{r}$, because the response constraint will override the delay constraint whenever $t_{\max} < 2r_{\mathrm{hi}}$.

6 Simulation Results

6.1 Simulation Specification

Simulations were run using **ns** [ns] on synthetic topologies. Every link is identical and symmetric:

propagation delay	10 ms
bandwidth	1.5 Mbps
queue limit	10

Each link carries background traffic in both directions that originates at one end and terminates at the other:

rate	90% of capacity
packet size	1088 bytes
interarrival time	exponential

The background traffic causes an observed loss rate of 2.2% of 1088-byte data packets, and an observed queuing delay with mean 19.8 ms and standard deviation 15.0 ms.

The source sends a 1088-byte data packet (including an update) every 100 ms. Requests are 64 bytes and responses are 1088 bytes.

The members use the following parameter values:

Loss detection threshhold multiplier: 0

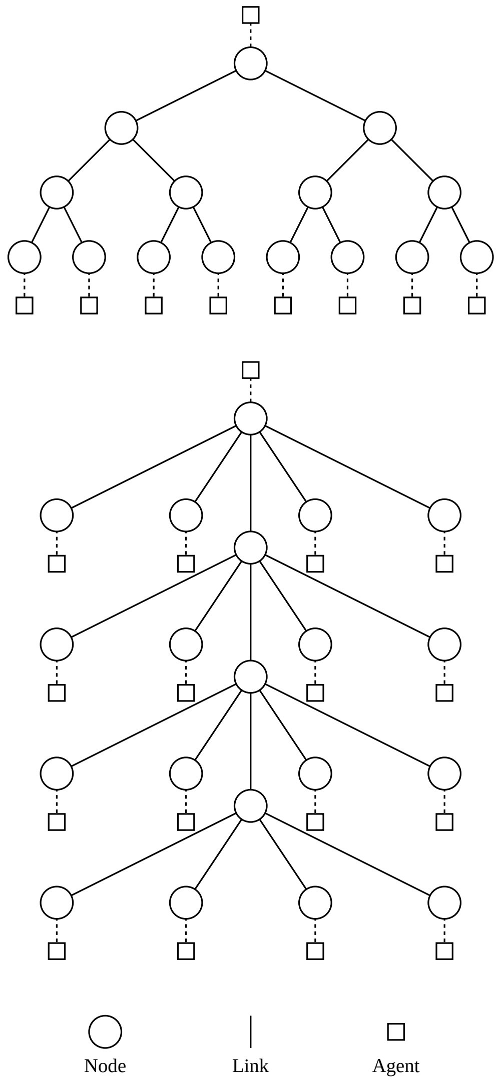


Figure 7: Topologies used in the simulations: balanced binary and square.

$\hat{r}$: exponential weighted moving average with weight $\frac{1}{8}$ for the mean and $\frac{1}{4}$ for the variance (like TCP).

r_{hi}: $\hat{r}$ mean plus 3.0 times $\hat{r}$ standard deviation.

Response suppression threshhold multiplier: 3.0

Default t_{req}: $now - r_{\mathrm{hi}}$

Null request rate: 0

Deletion timeout: infinite

For tuning μ, three behaviors were simulated, all using the unified tuning algorithm:

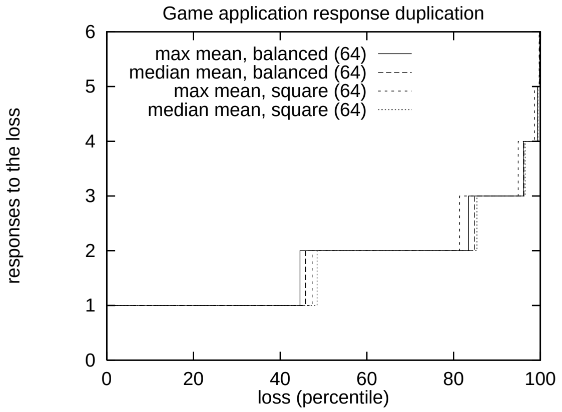


Figure 8: Duplicate response distribution.

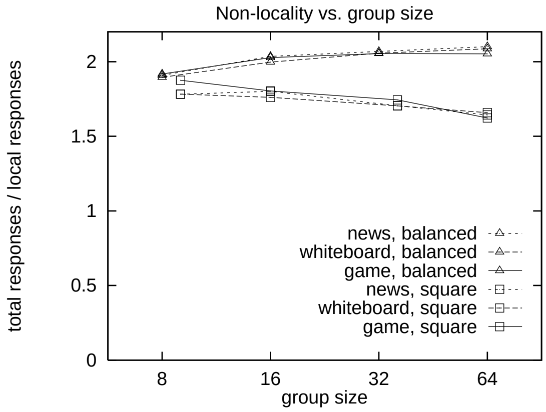


Figure 9: Exposure, averaged over all losses by dividing the total number of responses received by all members by the total number of local responses received by all members.

application	$t_{\max}$	p	$x_{\max}$
news	600 s	10^{-9}	2.0
whiteboard	5 s	10%	4.0
game	0.2 s	1%	4.0

Two topologies were used, a balanced binary tree and an unbalanced square tree (figure 7), with sizes varying between 8 and 64 members (**ns** could not handle 128).

Every run lasts 15 minutes of simulated time, during which 9000 data packets are sent.

6.2 Response Duplication

Each curve of figure 8 shows, for the game application with 64 members, the number of responses received by one member for each loss it experienced. The losses are situated along the horizontal axis, sorted by number of responses. The number of responses per loss varies between 1 and 7, with a mean of about 1.8, well below the $x_{\max}$ constraint of 4.0 (the application used the extra-pessimistic prediction, which did not come true). There are two members shown for each topology: the one with the maximum mean responses per loss, and the one with the median mean. The maximum and median are very close, and topology makes very little difference. The curves for the whiteboard and news applications (not shown) are similar, but shifted to the right. The whiteboard application got 1 to 4 responses per loss with a mean of 1.2 to 1.3, while the news application got 1 to 3 responses per loss with a mean of 1.1 to 1.2.

6.3 Request Traffic

The analysis predicts that members should receive, on average, as many local requests as local responses (where "local" means pertaining to a loss that the member experienced), and that the requests should not be concentrated at a few members. For the balanced topology, the ratio of local requests to local responses received by any single member (over all losses) varied between 0.4 and 1.1, depending on the application and group size. For the square topology, the ratio varied between 0 and 1.4, depending on the application, group size, and position in the tree.

6.4 Response Locality

The analysis predicts that exposure (non-locality) depends on topology, but at worst grows as $O(\log N)$. Figure 9 shows exposure versus group size for all three applications and both topologies. The application makes little difference. The balanced topology shows a modest increase with $\log N$, while the square topology shows a decrease, possibly because as it grows it becomes less balanced.

6.5 Retransmission Delay

The application with a delay constraint is the whiteboard. Each curve of figure 10 shows, for a group size of 64, the retransmission delay for each loss experienced by one member. The losses are situated

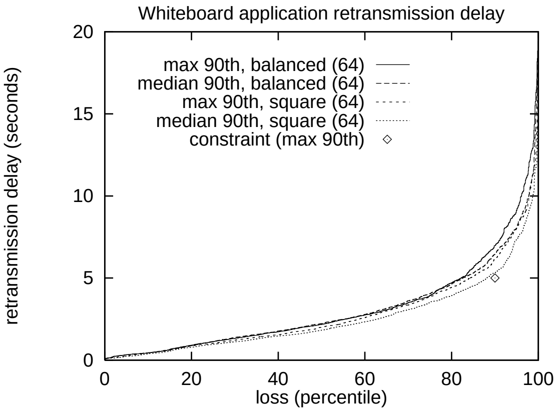


Figure 10: Retransmission delay distribution.

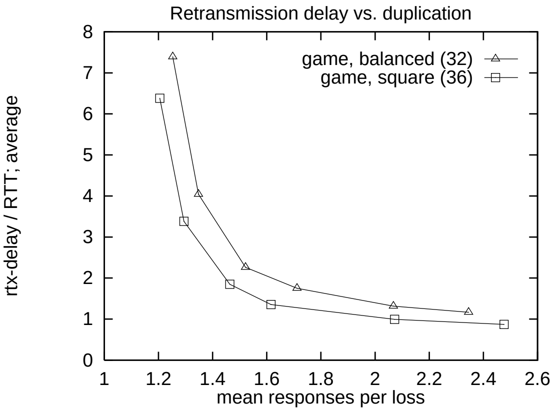


Figure 12: Delay/overhead trade-off.

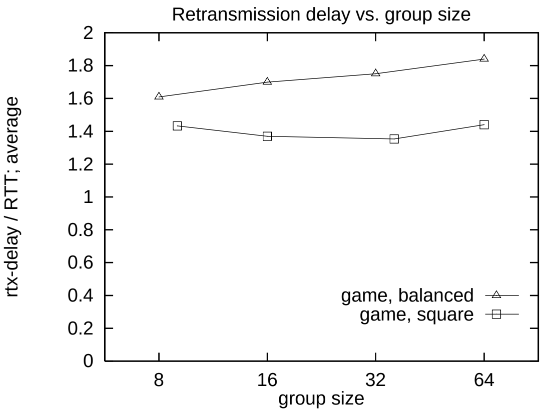


Figure 11: Normalized retransmission delay, averaged over every loss experienced by every member.

along the horizontal axis, sorted by delay. The analysis predicts the shape of the curves (exponential) and that they should all satisfy the delay constraint (10% chance of exceeding 5 s) by passing through the indicated point. There are two members shown for each topology: the one with the maximum delay at the 90^{th} percentile, and the one with the median. The maximum and median are very close, and the topology makes little difference. The small deviation of the curves from the prediction is presumably due to the simplifying assumptions, particularly the neglect of the analysis to account for lost requests and responses.

The analysis predicts that retransmission delay is independent of N, but dependent on the response time r. Because response time grows as the topology scales up, let us *normalize* the retransmission delay by dividing by the mean round trip time between the member and the source. Figure 11 shows normalized retransmission delay versus group size for both topologies for the game application, the only one that attempts to maintain a constant overhead. The analysis predicts horizontal curves. The balanced topology shows a small increase with group size, presumably because paths lengthen as the topology grows, increasing the fraction of requests and responses that are lost, which the analysis does not account for. The square topology shows smaller delays, presumably because the average distance between members is smaller compared to the height.

Figure 12 shows the trade-off the game application can make between delay and overhead by choosing different values of $x_{\max}$. The third point from the right on each curve corresponds to $x_{\max} = 4.0$ as in the other graphs. For the balanced tree, this operating point gets 1.7 responses per loss compared to 1 for LMS (by construction), and the normalized retransmission delay is about 1.8 compared to 0.9 for LMS, so it is within a factor of 2 of LMS on both axes simultaneously. The unbalanced topology performs better than the balanced one on both axes, but we lack results for LMS on this topology.

7 Future Work

7.1 More Simulations

In order to simulate large groups, **ns** must be used more efficiently, or another simulator must be used.

Also, simulations using real topologies should be conducted.

7.2 Shared Trees

This paper has focused on source-rooted trees, but in practice IP multicast will also use shared trees, in which the source is not the root of the distribution tree, but is rather a leaf[7]. If we can arrange for the root of the tree, when it decides to forward a randomcast packet "upward", to tunnel the packet to the source, then the search party approach should still work. For all edges except those on the path between the source and root, "toward the source" and "toward the root" are the same, so most recovery behavior will be unaffected. When a loss occurs on the way from the source to the root, there will not be a loss subtree, but rather a *non-loss subtree*. If the non-loss subtree is small compared to the whole tree, a request will soon be tunneled from the root to the source, who will send a response to the entire tree. If the non-loss subtree accounts for much of the tree, the recovery might involve multiple requests entering the non-loss subtree and generating responses that carry the data out to various separate subtrees. In either case, Search Party should work unmodified on shared trees, but simulations are needed to test this hypothesis.

Alternately, if routers maintain positional labels as proposed in [LLG96], they can "re-hang" the shared tree to make any source appear to be the root for packets carrying the positional label of the source.

7.3 Rumor Mill

Search Party is not the only way to use randomized forwarding for scalable multicast loss recovery. Another new approach, called Rumor Mill, is much simpler, though it has longer retransmission delays. After detecting a loss, a member sends requests via *trivial randomcast* (uniform distribution upward and downward with nothing inserted into the packet, so routers need not know subtree populations nor even which way is "up") with mean rate μ until a response arrives. If a responder has the requested data, it *unicasts* a response to the requestor, so subcast is not needed. The loss subtree does not get the retransmission all at once; rather, the data spreads iteratively, with the repaired population doubling every so often.

This approach has perfect locality because a member will never get a response if it sends no requests. It has a tunable trade-off between duplicate responses and retransmission delay like Search Party. The delay is increased by a factor of $O(\log L)$. Rumor Mill does not lend itself to analysis, so simulations are needed.

One advantage of this approach that may appeal to internet service providers is that only the source, not the members, ever sends a packet to multiple hosts. Subcast enables any member to send to the group even if source-specific joins were used, but Rumor Mill does not use subcast.

8 Conclusion

Some applications need a reliable multicast service, which can be built on top of the unreliable IP multicast service. Scalability requires local recovery, but some approaches like RMTP have limited feasibility because they attempt to mirror the topology above the network layer without access to the routing information. LMS uses the actual topology, but has limited robustness because responsibility is concentrated. Robustness can be improved using randomization. The Search Party approach uses randomized forwarding to provide robust loss recovery while retaining scalability and performance.

Acknowledgments

Thanks go to Christos Papadopoulos and George Varghese for discussing LMS.

This research began as part of a class project with Eshwar Belani.

[7] This is the situation in core-based trees [BFC93]. In PIM sparse-mode [DEF+94] the source still appears to be the root, because packets go from the source to the rendezvous point before entering the tree.

References

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT) an architecture for scalable inter-domain multicast routing. In *Proceedings of SIGCOMM '93*, San Francisco, USA, August 1993.

[DC90] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems (TOCS)*, 8(2):85–110, May 1990.

[DEF+94] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An architecture for wide-area multicast routing. In *Proceedings of SIGCOMM '94*, London, U.K., September 1994.

[FJL+97] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997. http://www-nrg.ee.lbl.gov/floyd/srm-paper.html.

[HC] Hugh Holbrook and David Cheriton. EXPRESS multicast: An extended service model for globally scalable IP multicast. http://gregorio.stanford.edu/holbrook/express/.

[KP91] Phil Karn and Craig Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM Transactions on Computer Systems (TOCS)*, 9(4):364–373, November 1991.

[KRT+98] Satish Kumar, Pavlin Radoslavov, David Thaler, Cengiz Alaettinoglu, Deborah Estrin, and Mark Handley. The MASC/BGMP architecture for inter-domain multicast routing. In *Proceedings of SIGCOMM '98*, Vancouver, Canada, September 1998.

[LLG96] B. Levine, D. Lavo, and J.J. Garcia-Luna-Aceves. The case for concurrent multicast using shared ack trees. In *Proceedings of ACM Multimedia*, pages 365–376, November 1996.

[LP96] J. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *IEEE INFOCOM '96*, San Francisco, USA, March 1996.

[Mil95] David L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Transactions on Networking*, 3(3):245–254, June 1995. See also RFC 1305.

[MRBP98] A. Mankin, A. Romanow, S. Bradner, and V. Paxson. *RFC 2357: IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols*. http://www.rfc-editor.org/, June 1998.

[ns] UCB/LBNL/VINT network simulator **ns** (version 2). http://www-mash.cs.berkeley.edu/ns/.

[PPV98] Christos Papadopoulos, Guru Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *IEEE INFOCOM '98*, San Francisco, USA, March 1998. http://www.ccrc.wustl.edu/~christos/PostScriptDocs/Infocom98-final.ps.Z.

[Str89] Peggy Tang Strait. *A First Course in Probability and Statistics*. Harcourt Brace Javanovich, second edition, 1989.

A Proofs and Derivations

This section uses the notation introduced in sections 2 through 5, augmented by the use of L_i as an abbreviation for $L(X_i)$.

A.1 Randomcast Forwarding Statistics

Suppose there is one member M at node X_0. The probability that a randomcast packet sent by M successfully travels upward through nodes $X_1, X_2, X_3, \ldots$ to emerge upward from node X_i, is

$$\frac{L_0}{L_1}\frac{L_1}{L_2}\frac{L_2}{L_3}\cdots\frac{L_{i-1}}{L_i} = \frac{1}{L_i}$$

(Recall that $L_0 = 1$.)

If a subtree contains L leaves, and mL randomcast packets are sent from its leaves, they constitute mL independent Bernoulli trials, each with a $1/L$ probability of success (escaping). Therefore the probability of exactly k escapees is given by the binomial probability function

$$\begin{aligned} b(x; n, p) &= b(k; mL, 1/L) \\ &= \binom{mL}{k}\frac{1}{L^k}\left(1-\frac{1}{L}\right)^{mL-k} \end{aligned}$$

By Poisson's limit law [Str89], as n increases while np remains constant, $b(x; n, p)$ approaches the Poisson probability function $p(x; \lambda)$ with $\lambda = np$. Therefore the probability of exactly k escapees is

$$b(k; mL, 1/L) \approx p(k; m) = \frac{m^k}{k!e^m}$$

If the number of randomcast packets sent by each leaf has a Poisson distribution with mean m, then the number of escapees has a true Poisson distribution $p(k; m)$, not a binomial distribution.

A.2 Null Response Traffic

The null responses received by M using X_k as exploder are the ones corresponding to null requests that were forwarded upward from X_k then bounced downward by X_{k+1}. If all members use the same ρ, null request traffic is ρ on every upward link, so X_{k+1} bounces null requests from X_k at the rate $\left(1 - \frac{L_k}{L_{k+1}}\right)\rho$. The total null response traffic received by M is thus

$$\begin{aligned} \rho \quad \cdot \quad & \left[\left(1-\frac{L_0}{L_1}\right) + \left(1-\frac{L_1}{L_2}\right) + \left(1-\frac{L_2}{L_3}\right) + \ldots \right. \\ & \left. + \left(1-\frac{L_{n-1}}{L_n}\right) + 1\right] \end{aligned}$$

(The last 1 is for the source, who never forwards requests, but always responds to them.)

Consider the sum

$$S = \frac{L_0}{L_1} + \frac{L_1}{L_2} + \frac{L_2}{L_3} + \ldots + \frac{L_{n-1}}{L_n}$$

The product of the terms is L_0/L_n. It is well-known that the geometric mean of a set of n positive numbers is less than or equal to the arithmetic mean, which implies that the sum is greater than or equal to n times the n^{th} root of the product. Equality obviously holds when all the terms are equal. In our case, this implies that the minimum possible value of S is

$$n \cdot \left(\frac{L_0}{L_n}\right)^{1/n}$$

This in turn implies that the maximum null response traffic received by M is $\rho \cdot (n - S + 1) = \rho z$ where

$$z = n - n \cdot \left(\frac{L_0}{L_n}\right)^{1/n} + 1 = 1 + n \cdot \left(1 - e^{\frac{1}{n}\ln\frac{L_0}{L_n}}\right)$$

$$\text{let} \quad u = \frac{1}{n}\ln\frac{L_0}{L_n}$$

$$\frac{dz}{dn} = 1 + (u-1)e^u$$

Because $u < 0$, we have $\frac{dz}{dn} > 0$, so z is maximized as n approaches infinity. Expanding z using the Taylor series for the exponential function reveals that

$$\lim_{n\to\infty} z = 1 - \ln\frac{L_0}{L_n} = 1 + \ln\frac{L_n}{L_0} = 1 + \ln N$$

(Recall that $L_0 = 1$ and $L_n = N$.)

Using the same analysis to calculate response traffic using only X_i through $X_{j<n}$ as exploder, we would obtain $\rho \ln\frac{L_j}{L_i} = \rho \cdot (\ln L_j - \ln L_i)$. That is, the null response traffic from an interval of a member's ancestry is proportional to the difference between the balanced heights of the top and bottom of the interval, or in other words, null responses are distributed evenly over the balanced heights.

A.3 Recovery Performance

For analyzing performance during recovery, we assume that all members in the loss subtree detect the loss at time 0, that they all agree on $\hat{r}$, that the response time r is the same for every escapee, that a response arrives simultaneously at all members, and that requests and responses are not lost. Let μ denote the average μ value among the members of the loss subtree.

A.3.1 Number of Escapees

If $r \leq \hat{r}$, a member will know before $\hat{r}$ has elapsed whether any of the burst requests escaped, so the expected number of escapees is the expected number of burst requests that escape ($\hat{r}\mu$), plus the probability that none of them escape ($e^{-\hat{r}\mu}$) times the expected number of non-burst requests that escape. The first non-burst request to escape will produce a response that causes the requests to cease, so an average of $r\mu$ requests per member get sent after the one that escaped first, of which an expected $r\mu$ escape. Therefore the total expected number of escapees is

$$x(r) = \hat{r}\mu + (1 + r\mu)e^{-\hat{r}\mu}$$

If $r \geq \hat{r}$, the story is the same, except that in the case where one of the burst requests escapes (probability $1 - e^{-\hat{r}\mu}$) an extra $(r - \hat{r})\mu$ requests will be sent before the response comes back at time r. Adding this term to the previous result yields

$$\begin{aligned} x(r) &= \hat{r}\mu + e^{-\hat{r}\mu} + r\mu e^{-\hat{r}\mu} \\ &\quad + r\mu - \hat{r}\mu - r\mu e^{-\hat{r}\mu} + \hat{r}\mu e^{-\hat{r}\mu} \\ &= r\mu + (1 + \hat{r}\mu)e^{-\hat{r}\mu} \end{aligned}$$

If the network is not working at time 0, none of the burst requests can escape. There must be a first escapee, which was sent at time t_0, say. The corresponding response is received at time $t_0 + r$. In the interim, an extra $r\mu L$ requests are sent, of which an expected $r\mu$ escape. Therefore the expected number of escapees is

$$x(r) = 1 + r\mu$$

Because $(1 + u)e^{-u} \leq 1$ for $u \geq 0$, a network failure at time 0 makes $x(r)$ larger if $r \geq \hat{r}$. If $r \leq \hat{r}$, then the network failure does not necessarily increase $x(r)$; for example, if $r = 0$, the network failure always makes $x(r)$ smaller.

A.3.2 Upward Request Traffic

Let L be the number of members below any particular edge in the loss subtree. If a member M_i below this edge is sending requests at a rate μ_i, it contributes μ_i/L upward request traffic to the edge in question. The total upward request traffic on the edge is

$$\frac{\mu_1}{L} + \frac{\mu_2}{L} + \frac{\mu_3}{L} + \ldots + \frac{\mu_L}{L} = \frac{\mu_1 + \mu_2 + \mu_3 + \ldots + \mu_L}{L}$$

which is the average μ value below the edge.

A.3.3 Downward Request Traffic

If all members use the same μ, we can prove by induction that the downward request traffic is less than 2μ on any edge. It is certainly true of the edge where the loss occurred, since there is zero downward request traffic there. Any node with only one child passes all packets straight through in both directions, so the downward request traffic on the edge below is the same as that on the edge above. For a node with $c \geq 2$ children, suppose the request traffic arriving from above is less than 2μ. That traffic will be split evenly among the children, contributing less than μ to each. In addition, a child can draw at most $\frac{\mu}{c-1}$ request traffic from each of the other $c - 1$ children, so the total for each child is less than 2μ.

A.3.4 Exposure

Suppose a loss occurs just above node X_i. For each escapee, the number of members who receive the response will be L_i if X_{i+1} is the turning point, L_{i+1} if X_{i+2} is the turning point, and so on, up to L_{n-1} if X_n is the turning point, and $L_n = N$ if the request makes it all the way to the source (there is no turning point). Therefore the expected number of response-recipients per escapee is

$$\begin{aligned} &\left(1 - \frac{L_i}{L_{i+1}}\right) L_i + \frac{L_i}{L_{i+1}} \left(1 - \frac{L_{i+1}}{L_{i+2}}\right) L_{i+1} \\ &+ \frac{L_i}{L_{i+1}} \frac{L_{i+1}}{L_{i+2}} \left(1 - \frac{L_{i+2}}{L_{i+3}}\right) L_{i+2} + \ldots \\ &+ \frac{L_i}{L_{i+1}} \cdots \frac{L_{n-2}}{L_{n-1}} \left(1 - \frac{L_{n-1}}{L_n}\right) L_{n-1} \\ &+ \frac{L_i}{L_{i+1}} \cdots \frac{L_{n-1}}{L_n} L_n \end{aligned}$$

which simplifies to

$$L_i \quad \cdot \quad \left[1 - \frac{L_i}{L_{i+1}} + 1 - \frac{L_{i+1}}{L_{i+2}} + 1 - \frac{L_{i+2}}{L_{i+3}} + \dots \right.$$
$$\left. +1 - \frac{L_{n-1}}{L_n} + 1\right]$$

Using the same derivation as for null request traffic, we see that this expression cannot exceed $L_i \cdot \left(1 + \ln \frac{L_n}{L_i}\right) = L \cdot \left(1 + \ln \frac{N}{L}\right)$. Dividing by L yields the expected exposure.

A.3.5 Retransmission Delay

The probability $p(t)$ that no response has arrived before time t is the probability that no request sent before time $t - r$ escaped. For $t < r$, no requests were sent before time $t - r < 0$, so $p(t) = 0$. For $r \leq t \leq r + \hat{r}$, the number of requests sent before time $t - r \leq \hat{r}$ is $\hat{r}\mu$ per member, so $p(t) = e^{-\hat{r}\mu}$. For $t \geq r + \hat{r}$, the number of requests sent before time $t - r \geq \hat{r}$ is $(t - r)\mu$ per member on average, so $p(t) = e^{(r-t)\mu}$.

The expected value of the retransmission delay is

$$\begin{aligned} & \int_0^\infty t\, d\,(p(t)) \\ & = \quad r \cdot (1 - e^{-\hat{r}\mu}) + \int_{r+\hat{r}}^\infty t\, d\left(1 - e^{(r-t)\mu}\right) \\ & = \quad r + (\hat{r} + \tfrac{1}{\mu})e^{-\hat{r}\mu} \end{aligned}$$

(Hint: integrate by parts.)

A.3.6 Request Rate Selection

The end of section 5.5 suggests an algorithm for choosing μ based on application-specified parameters $t_{\max}$, p, and $x_{\max}$, and asserts that if $0 < p \leq e^{1-x_{\max}}$ and $x_{\max} > 1$ and $0 < r_{\mathrm{hi}} < t_{\max} < 2r_{\mathrm{hi}}$, then the response constraint overrides the delay constraint, which is to say

$$\frac{x_{\max} - 1}{r_{\mathrm{hi}}} \leq \frac{\ln p}{r_{\mathrm{hi}} - t_{\max}}$$

The first two prerequisities imply

$$0 < x_{\max} - 1 \leq -\ln p$$

The third prerequisite implies

$$r_{\mathrm{hi}} > t_{\max} - r_{\mathrm{hi}} > 0$$

We can divide to obtain the desired assertion.