Search Party: An Approach to Reliable Multicast with Local Recovery (work in progress)

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

14 Sep 1997

Abstract

Some applications, like software updates and distributed whiteboard, can benefit from a reliable multicast service, which can be built on top of the unreliable IP multicast service. Existing approaches have limited feasibility, scalability, or robustness. The new search party approach addresses these limitations. Analytic results appear promising enough to warrant simulation experiments.

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 to handle detection and retransmission of lost packets.

To forward multicast packets, IP routers use *distribution trees*. For each source/group pair, there is a tree in which the source is the root[1], the members are the leaves, the routers are the internal nodes[2], and the data links are the edges. When a packet arrives at a router from its parent, it forwards a copy to each child[3]. Whenever a packet is lost on a link, it fails to reach the members in the *loss subtree* (the subtree rooted at the node below the link).

If a reliable multicast protocol is to be scalable to large numbers of members, its first task is to avoid *implosion*. Reliable protocols involving a single sender and receiver invariably have the receiver send feedback to the sender, so the sender knows when it needs to send retransmissions. But if multiple receivers send feedback to a single sender, the

[1] More recently proposed multicast routing protocols like PIM [DEF+94] and CBT [BFC93] can use *shared trees*, in which the source is not the root, but this paper restricts its attention to source-rooted trees.

[2] A single host can play all three roles, but this paper treats them as logically distinct entities.

[3] Actually, the situation is more complex, because data links can have more than two endpoints. A router forwards a packet on an *interface* to a link, which may deliver it to multiple child hosts. This paper naïvely assumes point-to-point links to simplify the analysis. The real situation can be approximated in the naïve model by considering each multi-endpoint link to be a collection of point-to-point links between each pair of endpoints. Alternately, we can consider a multi-endpoint link to be a virtual router with a point-to-point link to each endpoint.

resulting implosion of packets can overwhelm the sender or the network links near it. The protocol's second task is to provide *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.

Section 2 summarizes some of the related work on reliable multicast. The search party approach is presented in section 3. Finally, plans for future work appear in section 4, conclusions in section 5, and acknowledgements in section 6.

2 Related Work

A good summary of reliable multicast work is included in [LLG96], which classifies reliable multicast protocols as sender-initiated, receiver-initiated with NAK avoidance (RINA), ring-based, and tree-based. In sender-initiated schemes, receivers send ACKs directly to the source, so they suffer from implosion. This section examines one tree-based protocol (RMTP), one RINA protocol (SRM), and a newer protocol that does not fit well into the taxonomy (unnamed—this paper calls it "the replier-based scheme").

2.1 RMTP

The Reliable Multicast Transport Protocol [LP96] organizes the members into a hierarchy. ACKs are sent not to the source, but to the parent member. Internal nodes in the hierarchy (called Designated Receivers) are able to send retransmissions from their data caches. RMTP therefore provides both implosion avoidance and local recovery. However, for RMTP to perform well, the hierarchy of members must be very closely correlated to the multicast distribution tree, and it is not clear how such a tree can be constructed, nor how it can track routing changes.

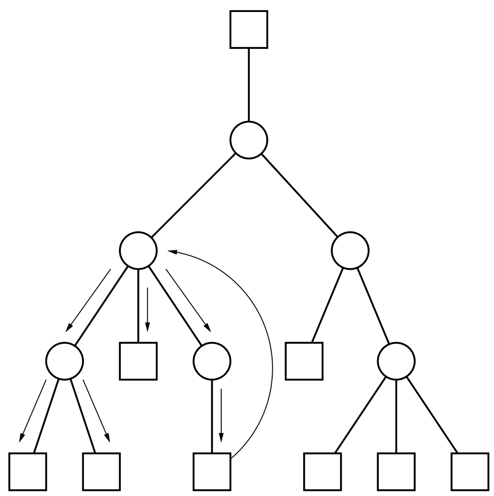

Figure 1: Subcast allows packet delivery to a subtree of the multicast distribution tree.

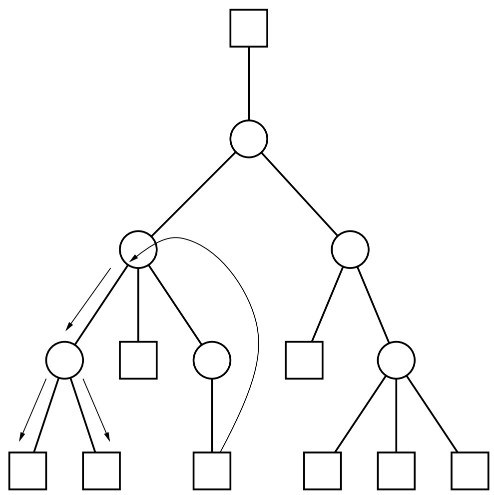

Figure 2: Directed multicast is similar to subcast, but further restricts the audience by selecting one child link of the subtree root.

A mechanism proposed in RMTP is *subtree multicast*, which defines a new IP packet type to let any host tunnel a multicast packet to a router, which allows the host to send a packet to an arbitrary subtree of a multicast tree. Various other implementations could be used to perform the same operation. This paper uses the term *subcast* to refer to any of them (see figure 1). The router at the root of the subtree is called the *exploder*.

2.2 SRM

In Scalable Reliable Multicast [FJL+97], 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. SRM therefore avoids implosion, but does not 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.

2.3 Replier

The replier-based scheme [PPV97] enlists the routers to provide a new (but simple) forwarding service. Each router in a multicast tree selects one of its child links to be the *replier link* (except that the router immediately below the source selects its parent link). Requests received from a non-replier link are forwarded to the replier link, while requests received from the replier link are forwarded to the parent (figure 3). Therefore, when every member in a loss subtree issues a request for the missing packet, exactly one request will escape the subtree, while the others go to members inside the loss subtree, who are unable to respond.

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 subcast packet, except that it also includes the link ID, which is used only by the turning point router to further restrict the audience (see figure 2).

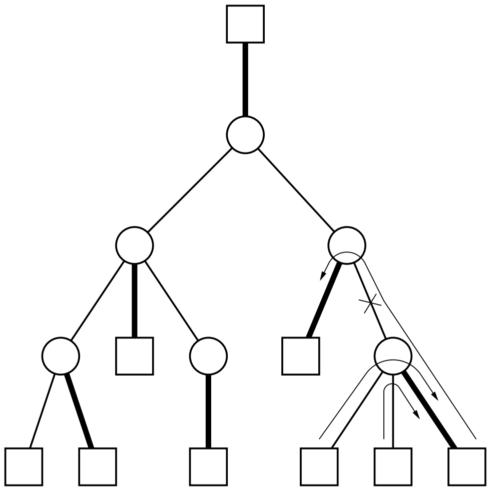

Figure 3: The paths taken by request packets after a packet loss in the replier-based scheme.

Notice that the new forwarding services required by the replier-based scheme are simple and do not violate layering—the routers do not need to see any transport-layer information like sequence numbers.

The replier-based scheme provides implosion avoidance and local recovery much like RMTP, but uses the multicast tree itself rather than a separate hierarchy, circumventing the question of how to construct and maintain the hierarchy. It avoids duplicate retransmissions without the use of random timers, so retransmissions are sent quickly, in one round-trip time. One potential drawback compared to SRM is fragility—the deterministic routing of requests to a single replier raises the risk that a single malfunctioning member or link could drastically affect a large number of other members.

2.4 Deletion

An issue that is addressed more carefully in some schemes than others is deletion. When is it safe for a host to delete old data? After all members have acknowledged it? What if a member leaves the group, or a new member joins? The semantics could get tricky, but in practice a very simple policy may suffice: a course-grain timeout. Let the source and every member hold on to data for a well-known amount of time before discarding it. Five minutes might be appropriate for some applications, a week for others, and the number can be announced the same way the multicast address is announced. If it takes longer than this amount of time to get a retransmission, 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.

3 Search Party

3.1 Overview

This section presents the search party approach, a reliable multicast protocol built on top of *randomcast*, a new forwarding service that routes packets randomly inside a multicast distribution tree. This approach inherits the structure of the replier-based scheme: requests are sent upward and bounced back down by a turning point router, and responses are sent via directed multicast. By randomizing the routing of requests, the search party approach trades away a little bit of performance (a constant factor) in exchange for increased robustness: a malfunctioning member has a small effect 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. A side effect of the randomization is that applications with varying delay sensitivities are able to fine-tune the trade-off between retransmission delay and recovery overhead.

When a member detects a loss, it begins sending requests that get routed randomly, and stops when a retransmission arrives. The member can be considered to be conducting 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, as will be shown in more detail.

Although the search party approach, like the replier-based scheme, relies on a new forwarding service for requests, there are ways for the endpoints to emulate randomcast forwarding themselves (approximately), which would ease deployment. The emulation is called *virtual randomcast*. Subcast would still be needed, but it is an extremely simple mechanism requiring no extra state in the routers.

Section 3.2 describes randomcast forwarding, and section 3.3 explains the reliable multicast protocol built upon it. Performance is discussed in section 3.4, and virtual randomcast in section 3.5. Throughout these sections, quantitative claims are made. The proofs and derivations appear in appendix A.

3.2 Randomcast Forwarding

At the heart of the approach is the new packet forwarding service, *randomcast*. Like a multicast packet, a randomcast packet identifies a multicast tree by a multicast group address and a root address. When a randomcast packet arrives at a router, the router may forward it to any one of its neighbors in the tree except the one the packet came from. Whenever a router acts as a turning point (receives a randomcast packet from a child and forwards it to another child), it inserts into the packet its own address and an identifier for the arrival link.

The probability distribution used to select the outgoing link[4] is critical to the behavior of systems using the randomcast service. Two distributions were considered: uniform, and weighted by subtree size. The distribution used for choosing between the parent and the children need not be the same one used for choosing among children.

3.2.1 Parent or Child?

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

Because routers already need to exchange messages with their children in order to maintain the tree, adding information to these messages would allow routers to keep track of the number of leaves below themselves and their children. This modest burden, and the increased router state, is avoided if the uniform distribution is used. One advantage of the weighted distribution is that it eases analysis, because the probability that a randomcast packet

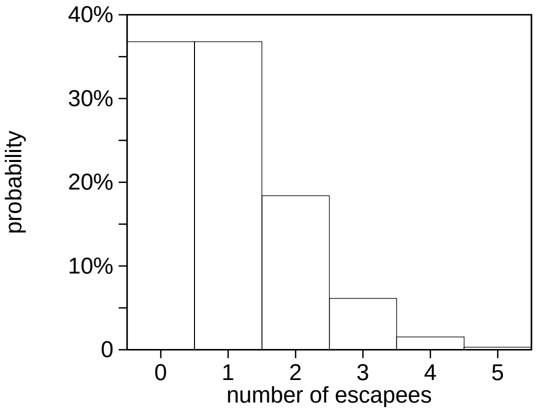

Figure 4: Probability that k randomcast packets escape a large subtree in which each leaf sends one packet (Poisson distribution with mean 1).

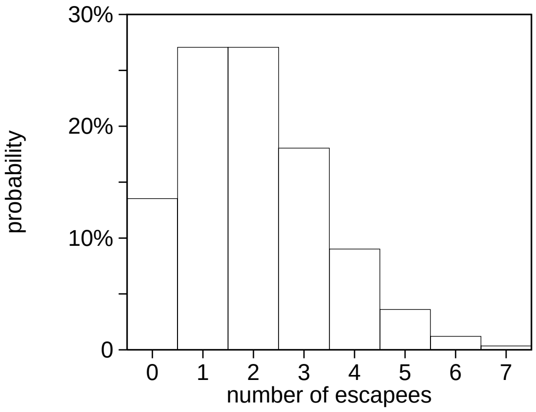

Figure 5: Probability that k randomcast packets escape a large subtree in which each leaf sends two packets (Poisson distribution with mean 2).

[4] Because we are assuming point-to-point links, there is no distiction between the outgoing link and the next node. In reality, there may be more than one neighbor on a single link. One way for routers to deal with this is to imagine a virtual link to each neighbor. This might introduce artifacts not accounted for by the analysis. A more correct approach is to imagine that each multi-endpoint link is a virtual router with point-to-point links to the endpoints. Each real router would have to perform the randomcast routing computations on behalf of its virtual children.

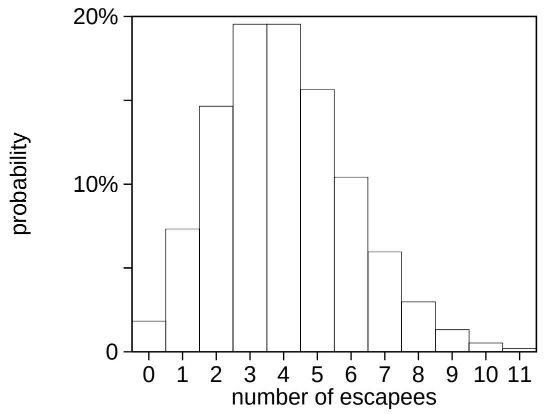

Figure 6: Probability that k randomcast packets escape a large subtree in which each leaf sends four packets (Poisson distribution with mean 4).

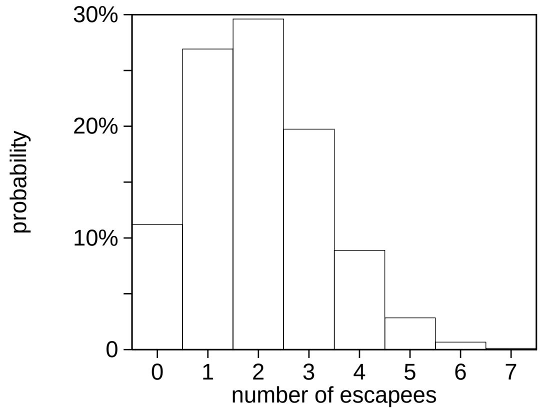

Figure 7: Probability that k randomcast packets escape a subtree of six leaves in which each leaf sends two packets (binomial distribution with $n = 12$ and $p = 1/6$). Compare this to figure 5.

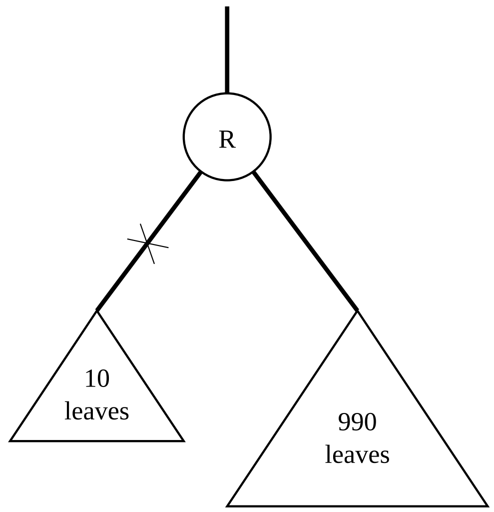

Figure 8: A packet loss in an imbalanced tree.

originating at any leaf below router R reaches R's parent is $1/L(R)$, regardless of the topology. It follows that if m randomcast packets are sent from each of the L leaves in a subtree, the number of packets to escape the subtree has a binomial distribution with $n = mL$ 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 figures 4 through 6. 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 7.

Simulations suggest that using the uniform distribution would not adversely affect the predictability of the flow of requests out of a subtree; however, it would have an adverse effect on *which* requests escaped.

Consider the scenario of figure 8. With a uniform distribution, there is a 50% chance that a request from the smaller subtree will be forwarded to R'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 R's parent, but the response will still be of interest to 99% of the recipients.

For the choice between parent and children, the search party approach uses the weighted distribution.

3.2.2 Which Child?

When a packet is to be forwarded by a router R to a child, either because it arrived from the parent or because the router 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 $L(C)/L(R)$ if the packet arrived from the parent. If the packet arrived from child A, the probability that it gets forwarded to child C is $L(C)/[L(R)-L(A)]$. It follows that before the router 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 $L(C)/L(R)$.

A router using the weighted distribution for selecting children will tend to distribute randomcast packets evenly among its descendent leaves, but packets from many incoming links can get concentrated onto one outgoing link. If that happens several times in a succession of routers, the downward traffic on a single link could grow exponentially. With the uniform distribution, a router distributes packets evenly among its child links, avoiding that hazzard. 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, the search party approach uses the uniform distribution for selecting children.

The distribution used to select children is more elastic than the distribution used to select between the parent and the children. The latter determines the height of the turning point, which determines whether the request escapes the loss subtree, and determines the scope of the response. If these are not chosen well, the scheme will not be scalable. On the other hand, the precise member chosen to send the response affects only the response time and the probability that a correctly functioning member and uncongested network path are chosen, which merely nudges the performance one way or another. Therefore, it is probably safe for routers to try optional strategies for gently skewing their child-selection distribution in favor of shorter or less congested paths.

3.3 Reliable Multicast Protocol

For simplicity, let us consider the data being transferred to be a sequence of packets identified by sequence numbers, though in practice it would be possible and desirable to use more complex structures and naming conventions. Also, let us assume that the source has exactly one child. If it has more than one child, it can logically divide itself into two entities, one of which emulates a router that serves as parent to the real child routers and as child to the other half of the source.

Members detect packet losses by observing skipped sequence numbers, so the source is obligated to send packets continually (the rate is a trade-off between overhead and delay). Null data packets may be used when there is no data to send. Because the network can reorder packets, each member must estimate the delay variance, which it can do if the source puts a timestamp in every data packet. We need not assume that the source clock and the member clock run at the same rate, because the member can estimate the drift and compensate for it. Each member chooses as a *loss detection threshhold* a multiple of the estimated standard deviation of the delay. After observing a skipped sequence number, 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.

After a member detects a loss, it acts as a *requestor*, emitting randomcast *request packets* as a Poisson process with mean rate[5] μ until it gets a corresponding *response packet*. Like the loss detection threshhold, μ is a trade-off between overhead and delay, and each member may choose its own value. The parameters effectively get averaged over all members in the loss subtree during recovery. Because each member belongs to multiple subtrees of widely varying sizes, "strategic voting" (exaggerating one's own preferences to compensate for the preferences of others) is unlikely to work well.

Any member who receives a request acts as a *responder*, and reacts to the request in one of three ways. If the responder has already discarded the requested data because it got too old (see section 2.4), the responder returns a response packet that indicates that the requestor is out of luck. If the responder has the requested data, it sends a response packet containing the data. In either case, the responder uses the address and link ID written into the request by the turning point router to send the response via directed multicast. In the third case, the responder has never had the requested data, and takes no action. When the source receives a request, it behaves the same way a member would, except that it uses regular (undirected) multicast (the request has no turning point).

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 for the requests it sends (both the mean and variance), for reasons to be given below. The best-guess estimate is called $\hat{r}$, and the estimate of the upper bound (at some confidence level) is called r_{hi}. To allow this estimation, every request contains a *time reference*, which is a pair of numbers: the sequence number of the most recently received original data packet (not a retransmission), and an offset, which is the number of source clock ticks between the reception of that data packet and the sending of the request (the member must convert from its own clock ticks if the clock rates can differ). Every response contains a time reference copied from the request. The purpose of the time reference is not merely to avoid retranmission ambiguity [KP91]; it provides a sample r value to every member who sees the response, not just the one who sent the request. The sample value is the time the response is received minus the time the referenced data packet was received (if it was received) minus the offset. Because the difference between the times two members receive the response is probably about the same as the difference between the times they received the referenced data packet, they will calculate approximately the same sample value.

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

One use of the response time estimates is to refine the request behavior (the earlier description was simplified). A request sent after a loss is detected, but before enough time has passed for a response to have arrived, might as well have been sent as soon as the loss was detected. According to the simplified behavior, each requestor would send $\hat{r}\mu$ requests spread over the first $\hat{r}$, but it should instead send them all immediately when the loss is detected, then wait for $\hat{r}$ before beginning the Poisson request 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 $\mathrm{frac}(\hat{r}\mu)$. This refined request strategy can decrease retransmission delay and redundant 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 refined strategy shouldn't make any significant difference to the aggregate behavior of the members, compared to the simplified strategy.

The response behavior should also be refined. 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. When the response arrives, several 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 redundant responses. To prevent this, each member remembers not only the time it received the original transmission of each data packet (which was already necessary for the response time estimation), but also the last time it received a retransmission of the data. If a responder has the requested data, and has seen a retransmission of the requested data, it should map the request's time reference to a moment t_{req} by adding the offset to the arrival time of the *referenced* data packet[6], and compare t_{req} to the arrival time t_{rtx} of the most recent retransmission of the *requested* data. If t_{req} does not succeed t_{rtx} by at least one loss detection threshhold, then the responder should ignore the request. We are again taking advantage of the fact that the difference between the times that two members receive a multicast packet is about the same for each packet, and we are using the loss detection threshhold to allow for some discrepancy.

Because there may not be enough losses to generate enough sample r values, each member may send null requests as a Poisson process with a 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, it is proportional to the actual height. The member should probably give greater weight to real responses compared to null responses for the purpose of calculating $\hat{r}$, but not for calculating r_{hi}. Like μ values, ρ values effectively get averaged: a member's choice of ρ affects itself most, and other members decreasingly with distance.

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

3.4 Performance Analysis

In order to choose values for μ and ρ, application programs need to know their own delay requirements, and the effects that these parameters and the response time estimates $\hat{r}$ and r_{hi} have on expected performance after a loss. In order to make the analysis tractable, let us assume that every member in the loss subtree detects the loss at the same time, called time 0, that every member in the loss subtree has the same value for $\hat{r}$, that the response time r is the same for every *escapee* (request that escapes) from the loss subtree, and that a response arrives simultaneously at all members. To the extent that these assumptions approximate the truth, the analytic results approximate the true performance.

3.4.1 Number of Escapees

Let μ denote the average μ value 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 refinement of the request behavior 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

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

This prediction is an especially safe upper bound on x, because 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}.

3.4.2 Request Traffic

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.

If the packetized request traffic is approximated by smooth fluid flow, then during the recovery each link in the loss subtree, and the loss link itself, carries upward request traffic equal to the average μ value among the members below itself. Above the loss link, 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 link 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 link is ρ upward and less than 2ρ downward.

3.4.3 Response Locality

For each escapee, a response packet is generated. The expected number of members who receive the response depends on the topology, but cannot exceed $L \cdot \left(1 + \ln \frac{N}{L}\right)$. Let us define the *exposure* of a loss[7] as the total number of retransmitted packets delivered divided by the number of leaves in the loss subtree. The ideal exposure would be one. The expected exposure for the worst-case topology is $x \cdot \left(1 + \ln \frac{N}{L}\right)$.

3.4.4 Response Time

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 refinement of the request behavior, which is questionable. Therefore, when choosing a μ value, applications should consider the probability distribution only for $t \geq r_{\text{hi}} + \hat{r}$. The best-guess prediction is thus

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

and the pessimistic prediction is

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

The average (expected value) retransmission delay 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 ($r = r_{\text{hi}}$) predictions. Of course, the delay cannot be predicted at all if the network is not working at time 0.

3.4.5 Examples of Tuning

To see how application programs might tune the protocol to suit their own needs, let us consider three hypothetical applications.

[7] This is a generalization of exposure as defined in [PPV97].

Software update Suppose this application can tolerate arbitrary delays, but wants a probability of less than one in a billion that a retransmission will take longer than the discard timeout t_{disc}, which we will suppose is ten minutes. Round-trip times of more than ten seconds are unheard-of in the internet, so for simplicity the application can let $r_{\text{hi}} = 10\,\text{s}$ and not bother to estimate it. The refinement of sending $\hat{r}\mu$ requests immediately and waiting $\hat{r}$ has a negligible benefit when μ is small, as it will turn out to be. Therefore, the application can forgo the refinement by letting $\hat{r} = 0$ and not bothering to estimate it. The pessimistic delay requirement is

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

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

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

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.

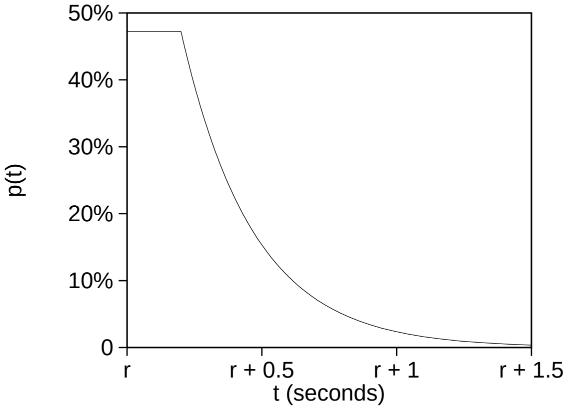

Figure 9: The probability $p(t)$ that no retransmission has yet arrived at time t for the multiplayer game application. Notice that the left edge of the graph is time r, not time 0.

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 pessimistic response requirement is

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

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 9 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 number of responses per loss, $x_{\max} > 1$. The response constraint overrides the delay constraint. The protocol library can then calculate

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

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 $t \geq r_{\text{hi}} + \hat{r}$, because the response constraint will override the delay constraint whenever $t_{\max} < 2r_{\text{hi}}$.

3.5 Virtual Randomcast

Subcast is a very simple forwarding service, requiring no additional state in the routers. Directed multicast is more complex. Randomcast is still more complex, and requires additional state in the routers. If the search party approach could be implemented using only subcast, it would be easier to deploy. *Virtual randomcast* is a technique that allows the members to emulate randomcast using only subcast.

If a member knows enough about the topology of the multicast tree, it can precompute the same random decisions that the routers would normally compute for a randomcast packet, determine the recipient, and send the packet directly there via unicast. How much must the member know? In order to emulate the routing up to the turning point, the member must know the number of leaves in each subtree to which it belongs, that is, it must know $L(R)$ for each router R in its ancestry. In order to fully emulate the routing after the turning point, the member would need to know the full topology, but as discussed at the end of section 3.2.2, the precise selection of recipient during the post-turning-point routing is not so important. As long as there are enough random leaves to choose from, and the selection is continually reshuffled over time, that should be sufficient.

3.5.1 Tree Estimation

The first thing every member needs to know is how many ancestors it has. If the source includes in the payload of every data packet a copy of the initial hop limit[8], the member can subtract the final hop limit to obtain the number of routers in its ancestry[9].

Let us focus attention on an arbitrary member M. Call M's parent router R_1, call R_1's parent R_2, and so on up to R_n, whose parent is the source. R_0 is an alias for M. For each R_i, M maintains a raw estimate ℓ_i of $L(R_i)$ (the number of leaves below R_i), and a corresponding cooked estimate

$$\hat{L}_i = \max_{0 \leq j \leq i} \ell_j$$

ℓ_0 is always 1. M also maintains per ancestor a queue Q_i containing addresses of members believed to be below R_i but not below R_{i-1}, and another queue q_i that is like Q_i but contains only very recent first-hand knowledge. Members maintain these estimates and queues by exchanging *census packets* via subcast.

M emits census packets as a Poisson process with mean rate ν. It chooses the exploder independently for each packet, as follows:

$k \leftarrow 0$
while $k < n$
  with probability $1 - \frac{\hat{L}_k}{\hat{L}_{k+1}}$ break
  increment k
endwhile
chose R_k

[8] Known as "time to live" in IPv4, this header field limits the number of times a packet may be forwarded. It is decremented by each router along the path.

[9] This result might need to be diminished by one or two, depending on how the endpoints treat the hop limit field.

If M chooses R_0 (itself), it sends an empty packet to itself, which can be optimized away. If M chooses R_n, it has the option of using regular multicast rather than subcast (remember from the beginning of section 3.3 that R_n must be the *only* child of the source).

A census packet using exploder R_k contains the depth D of R_k ($D = n - k$), the value of ν (converted to inverse source ticks, if the clocks do not all run at the same rate), the address of the sender, and k entries numbered 1 through k, where entry i contains $\hat{L}_i$, an identifier for R_i, and the contents of q_i.

The nature of the identifier depends on which of three styles of virtual randomcast is being used, which depends primarily on which of the possible subcast mechanisms is adopted. Two of the styles require members to know router addresses, whereas the third does not, at a cost in performance. The features common to all three styles are described first, followed by the specifics for each style.

In a census packet sent by M, entry i is understood to contribute $\hat{L}_i/\nu$ (a quantity of time) to R_i. Because of the way exploders are chosen, M emits census packets that mention R_i at a mean rate of $\nu/\hat{L}_i$. Therefore, the mean rate at which M contributes time to R_i is 1, which means that on average, M's contribution to each of its ancestors equals the elapsed time. Notice that this property does *not* depend on the accuracy of $\hat{L}_i$, nor on the particular value of ν.

All contributions to R_i are visible to M, modulo lost census packets. Because every member below R_i contributes to R_i at rate 1, M can compute its raw estimate ℓ_i by adding up the contributions over some time interval T and dividing by T. The length of the interval is a trade-off between the timeliness of the information and the variance of the estimate.

Whenever M receives a census packet (even one of its own), it determines which of its ancestors corresponds to entry k (the last entry), using either D or the identifier in entry k, depending on the style. Suppose entry k corresponds to $R_{k+\Delta}$. M then identifies the *nearest common ancestor* of itself and the sender of the packet, which is determined by the least j for which the identifiers in entries j through $k-1$ match those of $R_{j+\Delta}$ through $R_{k-1+\Delta}$, respectively[10]. Entries j through k contribute time to routers $R_{j+\Delta}$ through $R_{k+\Delta}$, respectively. M adds the sender address and all member addresses[11] appearing in entries 1 through $j-1$ to queue Q_j. Addresses are removed from the queue when they become too old. The age limit A is a trade-off between the timeliness of the information and the number of options available for delivering virtual randomcast packets. M also adds the sender address (if it is not M's own address) to queue q_j. The age limit for this queue is $1/\nu$, so it usually contains zero or one address.

3.5.2 Request/Response Handling

To send a virtual randomcast packet, M chooses an exploder R_k just as it would for a census packet. If $R_k = R_n$, the randomcast packet is sent to the source. Otherwise, M chooses a responder[12] randomly (uniformly[13]) from Q_{k+1}, inserts some infor-

[10] Alternately, the nearest common ancestor could be determined by the least j for which the identifier in entry j alone matches that of $R_{j+\Delta}$. When everyone has a consistent view of the topology, the two definitions are equivalent, but they behave differently during routing changes. The alternative has yet to be fully investigated, but preliminary thoughts suggest that the original definition is safer.

[11] For large groups, adding *all* the addresses can result in unnecessarily large queues. An easy fix is to maintain a exponentially weighted moving average of the rate at which addresses are arriving at each queue. Whenever an address is about to be enqueued, flip a weighted coin to decide whether to discard it instead. Choose the weight as a function of the arrival rate in order to keep the queue near a desired size, like 10.

[12] If Q_{k+1} is empty, M must revise its choice of exploder to R_{k+1} and try again. This is likely to happen only if $\hat{L}_k$ and $\hat{L}_{k+1}$ are about the same, which implies that these revisions are both rare and inexpensive.

[13] Until more research is done, a uniform distribution is assumed and recommended. One avenue for exploration is to include a time reference in every census packet. M could then calculate a sample round-trip time between the packet's sender and its exploder, which might serve as a basis for skew-

mation into the packet (the nature of which depends on the style), and unicasts the packet to the chosen responder. The calculation that M performs to choose the exploder R_k is the same one that real randomcast routers would have performed[14] to choose the turning point R_{k+1}.

The differences between the three styles of virtual randomcast lie in the mechanics of the subcast service, the nature of the router identifiers, the way in which router identifiers are learned, and the way in which response packets are sent.

First style The only way a subcast packet informs the network which router is the exploder is by using the IP address of that router. Members learn the IP addresses of their ancestors by relying on the source to use the Record Route option periodically in data packets, and these addresses are used in census packets as the router identifiers. Members use the identifier of the last entry to determine Δ, so D may be omitted from the census packets. When a virtual randomcast packet is sent, the exploder address is inserted into the packet, allowing the responder to send the response via subcast.

Second style A subcast packet may inform the network which router is the exploder using the router's IP address, or using a hop limit[15]. If a hop limit is used, the exploder inserts its IP address into the packet. Router addresses are again used as identifiers, but the Record Route option is not needed. Census packets are sent using a hop limit. A census packet containing depth D informs the recipients of the address of R_{n-D}. $\Delta = n - D - k$, and the identifier in the last entry is ignored. Requests and responses are handled just as in the first style.

Whenever a member sends a census packet using exploder R_k, it scans $R_{i<k}$ for the router R_j with the stalest identifier. If the identifier's age exceeds some threshhold, the member subcasts an *identifier packet* using R_j as the exploder, containing only the depth $n - j$. Identifier packets can at worst double the overhead of virtual randomcast compared to the first style.

Third style The simplest version of subcast is used: a hop limit always designates the exploder, who never modifies the packet. Router addresses are never used for anything; the identifiers are random numbers called *cookies*. The sender of a census packet uses a newly-generated cookie in the last entry, and pre-existing cookies in the other entries. As in the second style, $\Delta = n - D - k$, but this time the identifier in the last entry is used—it is added to a queue of identifiers for R_{n-D}. Identifiers are removed from the queues as they become too old. An identifier matches a router if it matches any identifier in the corresponding queue. Identifier packets are used just as in the second style, but each must contain a newly-generated cookie in addition to the depth. To improve resiliance to lost census and identifier packets, census packet entries could contain two (or more) identifiers, the newest and a middle-aged one, of which only one need match.

Requests and responses must be handled differently when router addresses are not used. Requests contain the depth D of the exploder, plus a flag indicating whether the responder should subcast the response itself, or unicast it back to the requestor (an idea proposed in [FJL$^+$97]). If the responder subcasts the response, it must do so one level too high, at depth $D - 1$, because the desired exploder is not an ancestor of the responder. If the responder unicasts the response back to the requestor, containing a copy of D, the requestor can then subcast it using the proper exploder. The requestor must choose between increasing the exposure by a factor of $\hat{L}_{k+1}/\hat{L}_k$ versus roughly doubling the retransmission delay. The choice could be made using a simple threshhold on the exposure increase, with a value of

ing the distribution.

[14] Except that real routers might be able to treat multi-endpoint links as virtual routers and perform randomcast routing on their behalf. In virtual randomcast, multi-endpoint links are seen as multiple point-to-point links.

[15] An idea suggested by Steve McCanne. Normal IP packets are discarded when the hop limit reaches zero, but the action could vary depending on the packet type.

1 for delay-insensitive applications, and higher values for increasingly delay-sensitive applications.

3.5.3 Choosing Parameters

Although the computation of the L values compensates for differences in ν, it is necessary for all members to use roughly the same ν, otherwise there is no way to choose a sensible averaging interval T. The distribution of the relative error of the estimates has been too difficult to analyze. If the $\hat{L}$ estimates were accurate, the number of census packets M receives during an interval T mentioning a particular R_i would have a Poisson distribution with mean νT, suggesting that ν and T should be chosen to make νT about 20. Simulations will offer more insight.

If all members use the same ν value, and if the $\hat{L}$ estimates were accurate, they could each expect to receive census packets at a rate between ν and $\nu \cdot (1 + \ln N)$, which would also be the maximum downward census traffic on any link. The upward census traffic on every link is ν.

As N grows, the factor of $1 + \ln N$ requires an increase in the rate of census packet arrivals or a decrease in ν, or some of both. Decreasing ν, in turn, requires an increase in T or a decrease in νT, or some of both. Since νT is less understood, let's not monkey with it for now. A simple policy would be to let $\nu = \nu_0 / \sqrt{1 + \ln \hat{N}}$ and $T = T_0 \sqrt{1 + \ln \hat{N}}$, where $\hat{N} = \hat{L}_n$, ν_0 is published (like the multicast group address), and T_0 can be chosen independently by each member, but is simply $20/\nu_0$ for now. For $\nu_0 = 0.5/\text{s}$, we get the following scaling:

$\hat{N}$	averaging interval T (s)	mean census packet inter-departure time $1/\nu$ (s)	mean census packet inter-arrival time $\frac{1}{(1+\ln \hat{N})\nu}$ (s)
20	79	4.0	1.00
100	95	4.7	0.84
1000	112	5.6	0.71
10^5	144	7.1	0.57
10^9	186	9.3	0.43

For $\hat{L}_n < 20$, members should use $\hat{N} = 20$ for calculating ν and T because they would be too sensitive to small fluctuations in $\hat{L}_n$ in that range. Since A ought to vary with N in the same way that T does, and since A and T both determine how long information from census packets should be retained, members should probably set $A = T$.

The request traffic on links cannot be analyzed as it was for real randomcast, because it is not confined to the distribution tree. The request traffic incident to a particular member is highly dependent on topology, but varies directly with $\frac{\mu}{\nu A}$. The worst case from the point of view of the member is a loss affecting the entire tree shortly after the member has sent a census packet to the entire tree. In a balanced b-ary tree, the rate of incoming request traffic would be about $b \cdot (1 + \ln \log_b N) \frac{\mu}{\nu A} = O(\log \log N)$. Of course, real distribution trees are not balanced, but they must tend to become more balanced as they get larger. (If that were not true, the internet itself could not scale.)

4 Future Work

A simulator is under construction that will allow further study of the search party approach, using both real and virtual randomcast. Topological information about the MBone will provide realistic networks for the simulation experiments.

This paper has focused exclusively 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. 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 links 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.

5 Conclusion

Some applications require a reliable multicast service, which can be built on top of the unreliable IP multicast service. Existing approaches have limited feasibility (RMTP), scalability (SRM), or robustness (replier-based scheme). The search party approach addresses these limitations, and mathematical analysis suggests that it is worthy of implementation in a simulator to study its performance.

6 Acknowledgments

Thanks go to Steve McCanne for suggesting this problem and providing much useful feedback on the ideas explored, and also to Christos Papadopoulos and George Varghese for discussing their replier-based scheme.

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

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, CA, 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. University College London.

[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*, 1997. To appear.

[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.

[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, CA, March 1996.

[PPV97] Christos Papadopoulos, Guru Parulkar, and George Varghese. An error control scheme for large-scale multicast applications, 1997. http://www.ccrc.wustl.edu/~christos/PostScriptDocs/Infocom98.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 throughout section 3, augmented by the use of L_i as an abbreviation for $L(R_i)$. We assume that randomcast routers use the weighted distribution to choose between parent and children, and the uniform distribution to choose among children.

A.1 Randomcast Forwarding Statistics

The probability that a randomcast packet, initially sent by a member M toward router R_1, successfully travels upward through routers $R_1, R_2, R_3, \ldots$ to emerge upward from router R_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 $L_0 = L(M) = 1$.)

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

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

A.2 Null Response Traffic

The null responses received by M using R_k as exploder are the ones corresponding to null requests that were forwarded upward from R_k then bounced downward by R_{k+1}. If all members use the same ρ, null request traffic is ρ on every upward link, so R_{k+1} bounces null requests from R_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, which 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 $L_0 = L(M) = 1$ and $L_n = N$.)

Using the same analysis to calculate response traffic using only R_i through $R_{j<n}$ 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 endpoints 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, and that a response arrives simultaneously at all members. 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 immediate requests escaped, so the expected number of escapees is the expected number of immediate requests that escape ($\hat{r}\mu$), plus the probability that none of them escape ($e^{-\hat{r}\mu}$) times the expected number of non-immediate requests that escape. The first non-immediate request to escape will produce a response that causes the requests to cease, so there are an extra $r\mu L$ requests that 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 immediate 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 immediate 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 link in the loss subtree (or the loss link itself). If a member M_i below this link is sending requests at a rate μ_i, it contributes μ_i/L upward request traffic to the link in question. The total upward request traffic on the link 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 link.

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

A.3.4 Exposure

Suppose a loss occurs just above router R_i. For each escapee, the number of members who receive the response will be L_i if R_{i+1} is the turning point, L_{i+1} if R_{i+2} is the turning point, and so on, up to L_{n-1} if R_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

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

which simplifies to

$$\begin{aligned}L_i \quad \cdot \quad &\left[1-\frac{L_i}{L_{i+1}}+1-\frac{L_{i+1}}{L_{i+2}}+1-\frac{L_{i+2}}{L_{i+3}}+\ldots\right.\\&\left.+1-\frac{L_{n-1}}{L_n}+1\right]\end{aligned}$$

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 $\hat{r}\mu L$, 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 $(t-r)\mu L$, so $p(t)=e^{(r-t)\mu}$.

The expected value of the retransmission delay is

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

(Hint: integrate by parts.)

A.3.6 Request Rate Selection

The end of section 3.4.5 suggests an algorithm for choosing μ based on application-specified parameters p and $t_{\max}$, and asserts that if $0<p\leq e^{1-x_{\max}}$ and $x_{\max}>1$ and $0<r_{\text{hi}}<t_{\max}<2r_{\text{hi}}$, then the response constraint overrides the delay constraint, which is to say

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

The first two prerequisities imply

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

The third prerequisite implies

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

We can divide to obtain the desired assertion.

A.4 Virtual Randomcast Overhead

A.4.1 Census Traffic

M receives a census packet mentioning R_i whenever any member below R_i sends a census packet using R_i or above as exploder. A member sends census packets as a Poisson process with mean rate ν, and chooses R_i or above as the exploder with probability $1/\hat{L}_i$, so the member contributes census packets mentioning R_i as a Poisson process with mean rate $\nu/\hat{L}_i$. The total census traffic mentioning R_i is the union of L_i of these Poisson processes, which is itself a Poisson process. If the $\hat{L}$ estimates were accurate, the mean rate of the union process would be ν.

Because of the way exploders are chosen, there is an exact analogy between upward-moving census packets and upward-moving null requests, and between downward-moving census packets and downward-moving null responses (if the $\hat{L}$ estimates are accurate). Just as members expect null response traffic at a rate between ρ and $\rho \cdot (1 + \ln N)$, they expect census traffic at a rate between ν and $\nu \cdot (1 + \ln N)$. Just as upward null request traffic is ρ on every link, upward census traffic is ν on every link.

A.4.2 Request Traffic

The rate at which a member receives requests during a recovery obviously varies directly with μ. It also varies inversely with νA because responders are chosen randomly, and the number of candidates varies directly with ν (the rate and which members announce their candidacy) and A (the amount of time candidates are remembered).

In a balanced b-ary tree, M enqueues a candidate responder into q_i whenever any member below R_i but not R_{i-1} sends a census packet using R_i or above as exploder, which happens at rate $\frac{b-1}{b}\nu$ (assuming the $\hat{L}$ estimates are accurate). Since candidates survive in q_i for time $1/\nu$, the average length of q_i is $\frac{b-1}{b}$. The number of candidates reported per census packet is $1 + (i-1)\frac{b-1}{b}$, which we can shrink to $i\frac{b-1}{b}$ to simply the calculation (this will only make M's incoming request traffic look larger). The average length of Q_i is $\frac{b-1}{b}\nu \cdot i\frac{b-1}{b} \cdot A$. Immediately after M sends a census packet to the entire tree, it will be a candidate responder for each of its ancestor routers. If a loss affecting the entire tree occurs, the members below R_i but not R_{i-1} will collectively choose responders from their queues Q_i at a rate $(b-1)\frac{b-1}{b}\mu$ (because the members below each sibling of R_{i-1} collectively send requests using R_i as exploder at rate $\frac{b-1}{b}\mu$). Therefore M receives requests from members below R_i but not R_{i-1} at a rate $\frac{b}{i}\frac{\mu}{\nu A}$. Summing over i from 1 through $n = \log_b N$ reveals that M's total incoming request traffic is less than $b \cdot (1 + \ln \log_b N)\frac{\mu}{\nu A}$ (using the fact that the sum of n terms of the harmonic series is less than $1 + \ln n$).