

# Organizing Multicast Receivers Deterministically by Packet-Loss Correlation

Brian Neil Levine<sup>1</sup>, Sanjoy Paul<sup>2</sup>, and J. J. Garcia-Luna-Aceves<sup>3</sup>

<sup>1</sup> *brian@cs.umass.edu*, Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003 USA

<sup>2</sup> *sanjoy@lucent.com*, Bell Laboratories, Holmdel, NJ 07733 USA

<sup>3</sup> *jj@cse.ucsc.edu*, University of California, Computer Engineering Dept., Santa Cruz, CA 95060 USA

Received: November 1998 / Accepted: August 2001

**Abstract.** The ability to trace multicast paths is currently available in the Internet by means of IGMP mtrace packets. We introduce Tracer, a protocol that organizes the receivers of a multicast group deterministically into a logical tree structure while maintaining exact packet-loss correlation for local error recovery, and without requiring any changes to existing multicast routing protocols. Tracer uses MTRACE packets in IGMP to allow a receiver host to obtain its path to the source of a multicast group. Receivers use the multicast path information to determine how to achieve local error recovery and effective congestion control. We compare the tracing approach with prior mechanisms that attempt local recovery. Results of measurements carried out over the CAIRN illustrate the fact that tracing multicast paths is an effective tool to organize receivers based on their packet-loss correlation.

## 1 Introduction

As support in the Internet for multicast, or one-to-many, communication among a group of hosts continues to grow, applications taking advantage of the savings of multicast transmissions increase in number. Such applications include server-initiated “push” technology, shared whiteboards and multimedia conferencing tools, distributed interactive simulation environments, distributed games, and distributed web caches. While many unicast-based networked applications have taken reliable data transmission for granted, reliable transmission of multicast data remains an unresolved issue.

What has prevented the wide-scale deployment of reliable multicast protocols is the *acknowledgment-implosion* problem: as the number of receivers expecting reliable transmission of data multicast from a source increases, the source and the network become overwhelmed when receivers directly contact the source with either negative acknowledgments (Nacks) requesting lost data and the

retransmissions that ensue during packet-loss, or positive acknowledgments (Acks) of correctly received data. Producing a reliable multicast protocol that scales well with the number of receivers, in terms of network traffic and the processing required of the source and receivers, has proven to be a challenge, as demonstrated by the number of approaches taken in the past (e.g., [21, 5, 29, 12, 11, 7, 19, 28, 14, 20, 9, 15, 27]). Moreover, as protocols for multicast error control are developed, mechanisms must also be developed for multicast congestion control [6, 17, 3], similar to those developed for such unicast reliable protocols as TCP [8].

One technique that has been used in the past by reliable multicast protocols is *local recovery*: designating one or more hosts other than the source (usually one of the receivers) as another source of retransmissions when the network resources permit it. For instance, the Scalable Reliable Multicast (SRM) [5] protocol allows any receiver to respond to a Nack, and the Reliable Multicast Transport Protocol (RMTP) [21] appoints “designated receivers” organized into a tree structure for sending retransmissions to portions of the receiver set. Organizing the receivers into a tree structure, where each receiver is responsible for a set number of other receivers, has been shown by an analytical model to be the most scalable choice among several methods [23, 13]. Intuitively, reliable multicast protocols that organize trees of receivers for local recovery work well because they distribute the cost of processing Acks, Nacks, and retransmissions, which reduces the load on the source and the network. The question is then how can an organized local recovery hierarchy be established efficiently and what is a good topology for such a hierarchy.

We present a method of organizing multicast receivers which deterministically ensures Acks and Nacks are always sent to the best receivers in terms of packet-loss correlation or a combination of performance considerations. We show that such an organization is useful not only for multicast error control, but also for several proposals for multicast congestion control.

Section 2 discusses past techniques used to organize the receiver set for local recovery, and considers the lim-

\* The work at UCSC was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant F19628-96-C-0038.

itations of using each technique alone and in combination. Section 3 introduces a novel approach for organizing receivers—tracing the path from the source—heretofore overlooked as an immediately available tool built into multicast routers. Our protocol for organizing the receiver set, called *Tracer*, is based on tracing as a prediction of packet-loss correlation. We view *Tracer* as a *protocol component* for use in reliable multicast or multicast congestion control protocols. Section 3 also discusses small *local* improvements to routers that would allow subtree multicasting, or *subcasting*, which would greatly improve the efficiency of reliable multicast protocols, and shows why *Tracer* is particularly poised to take advantage of these improvements. Receiver organization with *Tracer* does not require subcasting. Section 4 details how *Tracer* can be used as a component of larger multicast protocols. In particular, we consider the Reliable Multicast Transport Protocol (RMTP) [21], the Structure-Oriented Resilient Multicast Protocol (STORM) [28], and De Lucia and Obraczka’s multicast congestion control scheme [3] as examples. Section 5 analyzes data collected on packet loss between several hosts on the Collaborative Advanced Internet Research Network (CAIRN), a real network, in order to illustrate how well tracing assists in predicting packet-loss correlation, and therefore receiver organization. Section 6 presents our conclusions.

## 2 Motivation

Much research in multicast error-control has converged to the notion of local aggregation of feedback and local recovery when the network permits it, which implies the definition of local groups of receivers. Current research in congestion control also seems to require a notion of groups, such that all members within a group have the same state of congestion or packet-loss correlation. The dynamic organization of group members is a crucial component of both error control and congestion control, and several methods have been used in the past for both approaches. These include measuring propagation delay or hop counts, exchanging packet-loss information, and providing router support at multicast routers. In this section, we consider each method and their limitations or why they may not be applicable, even in combination.

For clarity, we refer to receivers in error-control protocols sending Acks or Nacks as *requesters* and receivers that may respond with retransmissions as *retransmitters*. The set of retransmitters does not exclude the source. We may also think of a *parent* and *child* relationship between retransmitter and requester. The rest of this section examines existing approaches to grouping receivers while trying to preserve packet-loss correlation.

### 2.1 Measuring hop count

One of the main goals of selecting a retransmitter for each requester of a reliable multicast protocol is minimizing the delay in receiving retransmissions. For example, by measuring the number of routers on the path between

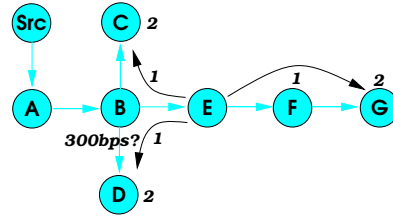


Fig. 1. Problems with using hop count.

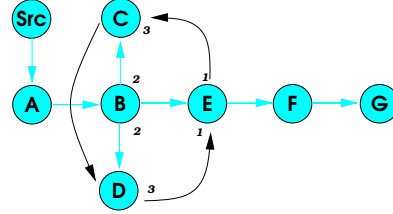


Fig. 2. Possible looping from using hop counts.

two hosts, i.e., the *hop count*, it is thought that the closest retransmitter (and if possible a retransmitter in the same administrative domain as the requester) can be located. For example, the Tree-based Multicast Transport Protocol (TMTP) [29] builds a tree of receivers by finding the closest retransmitter to each receiver based on hop counts between hosts. Figures 1 and 2 illustrate a multicast routing tree and three problems that can arise by using hop count as a measure in the way it is used in TMTP. First, in Figure 1, consider a host attached to router *E* which must choose between two retransmitters attached to routers *C* and *G*: both are the same number of hops away. The host attached to *C* is the better choice because data received at *G* must have passed through *E* already, and also because *C* is closer to the source than *G*, and generally that means there is less chance for packet loss. Furthermore, the host at *C* will probably detect the loss before *G* because it is closer to the source and expects the packet sooner; the host at *C* may then request the retransmissions before the host at *G*, and be able to service the host at *E*’s request sooner. However, using hop count as a metric in this manner cannot determine whether a host at *C* is a better retransmitter than a host at *G*.

Figure 1 also illustrates a second problem that may arise in using hop counts to select retransmitters. The host at *E* must choose between two retransmitters attached to routers *C* and *D*. Router *D* may lie behind a 300-baud modem, but such information is not available by hop counts (but may be revealed by propagation delay or packet-loss statistics).

Finally, using hop count as the only measure of packet-loss correlation can be subject to a looping condition, as illustrated by Figure 2. A host attached to router *C* may choose a retransmitter attached to router *D*; the same host at *D* may choose a retransmitter attached to router *E*. A loop of retransmitters forms if the host at *E* chooses the host at *C* as its retransmitter. If the hop count from the source is used, rather than between nodes, then the loop could be prevented, for example, if

nodes were required to find retransmitters that lie closer to the source than themselves.

The Reliable Multicast Transport Protocol (RMTP) [21] also builds a tree of retransmitters. Each receiver selects the closest retransmitter based on propagation delay. However, the tree of retransmitters must be picked by administrators to ensure that a loop-free hierarchy of retransmitters is formed.

In summary, retransmitters chosen by hop count alone do not provide any sense of relative positioning between a group of nodes, can result in loops if used incorrectly, and do not provide any information regarding link speed.

## 2.2 Measuring propagation delay

Another method used to reduce the delay in receiving a retransmission is to send requests to the closest node in terms of propagation delay: the smaller the propagation delay to the retransmitter, the sooner the retransmission will arrive. There are a number of caveats for using propagation delay, which we discuss here.

First of all, propagation delays on the Internet can be very dynamic and should only be used as an estimate, not as an exact measure of distance between nodes.

Second, the network resources used in measuring delay must be considered. Measuring the delay between two arbitrary hosts does not cost much by itself; however, if a protocol requires *every* host to know the propagation delay between itself and the source, the cost becomes unscalable with the number of hosts. For example, SRM requires that all receivers measure the roundtrip delay between each other. Even if hierarchical structures are introduced in the exchange of session messages with delay information, the accuracy of such information degrades as the scalability of session messages is increased by aggregating delay information [24]. Alternatively, the clocks of all hosts in the session could be synchronized, e.g., by using a protocol such as the Network Time Protocol (NTP) [16]; however, propagation delay between hosts must still be measured.

Finally, the unicast path between two hosts is not necessarily the same as the multicast path between the same hosts. Not all Internet routers are capable of routing multicast packets. This may be the case for a long time in the future because even as the multicast backbone increases many network domain administrators may enforce a policy where multicast traffic is forbidden to pass through certain routers. These paths may also differ because multicast routing protocols construct trees based on the paths between the source or core and the receiver set, rather than the paths between receivers. Therefore, measuring propagation delay between two hosts via unicast paths does not accurately measure the delay between the two on a multicast tree. If a protocol depends on delay to give relative positions on the multicast tree, then the protocol must measure multicast propagation delay in order to find the closest server. In other words, if unicast propagation delay is the measure, unicast retransmissions must be sent (as

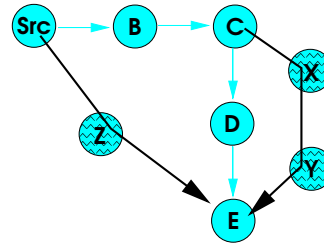


Fig. 3. Measuring multicast distances incorrectly with unicast propagation delay. Multicast paths are shown with gray arrows.

in STORM [28]). Similarly, if multicast retransmissions are to be sent from the closest server, then the multicast propagation delays must be measured (as in SRM [5]).

If a unicast-measured retransmitter uses multicasts to send the retransmissions, then the retransmitter might not be the closest host on the multicast tree, or even one that lies between the requester and the source (in other words, even the source may have been a better choice). Figure 3 illustrates that using a roundtrip unicast propagation delay can lead to choosing by mistake the farthest host on the multicast tree as the closest. Notice that the unicast path from the router attached to the source to host at router *E* (through router *Z*) is shorter than the unicast path from host attached to the router at *C* to router *E* (through routers *X* and *Y*). However, retransmissions multicast from *C* will reach *E* before those multicast from the source's router do, and unicast measurements will not detect this possibility.

Protocols that utilize multicast advertisements to measure roundtrip times or hop count must rely on shared multicast trees between sources. Otherwise, on per-source multicast routing trees, the advertisements used to measure round trip times are measured on a different trees, leading to inaccurate results. For example, both SRM and SHARQFEC measures roundtrip times based on multicast session announcements, and therefore would only work correctly on a shared multicast routing tree.

### 2.2.1 Multicasting or unicasting retransmissions

Regardless of what metric is used to choose the retransmitter, deciding between unicast retransmissions and multicast transmissions introduces its own set of problems. Unicast retransmissions trickle down the tree from retransmitter to retransmitter (e.g., the STORM protocol unicasts retransmissions despite its emphasis on deadline-oriented data); multicast retransmissions have been shown to lower average packet delay from reliable transmission [22]. Furthermore, unicast retransmissions are a waste of bandwidth; sending multiple copies of the same data is inefficient, especially since a multicast routing tree already connects the receivers. To make up for the delay from unicast retransmission, STORM allows nodes to change parents if the delay in receiving the retransmissions is too long, even though a parent may be waiting for the retransmission from a grandparent. As congestion and packet loss increase, the organization of

receivers may degrade into a scenario in which every host sends Nacks directly to the source.

On the other hand, multicasting retransmissions can also waste bandwidth and processing resources if the packets reach receivers that do not need the retransmitted data. Therefore, multicast retransmissions may save resources, but only if the receivers that require the data are the only recipients. Unfortunately, restricting the scope of a multicast packet using existing multicast technology can only be achieved by either forming and maintaining a new multicast group (which is costly), or by reducing the number of routers a packet may traverse by setting the *time-to-live* (TTL) field present in all IP packets to an explicit limit. Reducing the TTL of a multicast packet is crude as it lacks direction, i.e., packets are still disseminated in all directions from the retransmitter.

Another issue with multicasting retransmissions is that only one retransmitter should multicast data to the same set of receivers. However, this is not the case in SRM, which allows any receiver to answer a request after waiting for a backoff interval to check if other hosts have already acted. The advantage of multicasting retransmissions, as in SRM, is that some hosts may get retransmissions before they request them. However, because SRM uses a probabilistic approach, multiple retransmitters may act on a request.

In summary, multicasting retransmissions can be more efficient than unicasting retransmissions, provided that only one retransmitter responds, and that the scope of the retransmissions is only to receivers that require the data. This calls for a deterministic approach to multicast retransmissions, as well as the ability to accomplish subcasting, i.e., multicasting to a specific subset of a multicast group.

### 2.3 Using router support

Both the reliable multicast support provided by Addressable Internet Multicast (AIM) [12] and the Lightweight Multicast Scheme (LMS) [20] organize the receiver set of a multicast session into a tree by adding extra functionality to multicast routers. Rather than storing data at the routers of a multicast tree, the two protocols actually provide special routing support that can be exploited for the distribution of acknowledgments and retransmissions needed in reliable multicast protocols.

AIM and LMS have two advantages over approaches that use end-to-end techniques: the receivers' Nacks and Acks are automatically routed to a nearby receiver in a loop-free tree structure, and retransmissions can be sent to a subtree of the primary multicast tree, which is quicker and more efficient. AIM and LMS do not require expanding ring searches to coordinate the receivers of the session, which virtually all reliable multicast protocols do date use.

While the approaches taken by AIM and LMS are very similar, the differences lie in their operation. We detail the mechanisms used in AIM briefly because Tracer

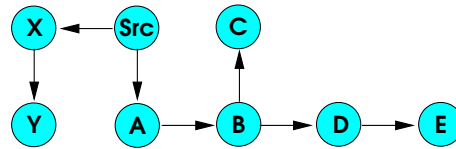


Fig. 4. For a host attached at router E, host attached to router D are directly acceptable; hosts at router C are indirectly acceptable; and hosts at router X are unacceptable.

specifically adapts the mechanisms used in AIM. The differences in operation between AIM and LMS mainly concern issues that Tracer solves in other ways, and are not of concern here; for example, working over shared-tree multicast protocols (such as CBT and OCBT [1, 25]), and avoiding sending redundant retransmissions to some receivers.

AIM is built on the concept we call *acceptability*, which considers whether a retransmitter is a good choice for a requester based on what percentage of the path they share from the source. If two hosts share a common path on the routing tree, then there exists a packet-loss correlation between the two hosts for any packets lost on the common path (see Section 5). More formally, the relationship between a retransmitter and a requester is either *unacceptable* (they share no common path from the source), or one of the following [12]:

Directly acceptable: The relationship between a receiver *A* and its retransmitter *B* is directly acceptable if the router attached to *B* lies on the path from the source to the router attached to *A*.

Indirectly acceptable: The relationship between a receiver *A* and its retransmitter *B* is indirectly acceptable if the routers attached to *B* and *A* share a common path from the source, and the router attached to *B* is closer to the source than the router attached to *A*.

Figure 4 illustrates all three cases. For a host attached at router *E*, hosts attached to router *D* are directly acceptable retransmitters because packets must travel through *D* before reaching *E*. For the same hosts attached to router *E*, hosts at router *C* are indirectly acceptable retransmitters because a common path (Src  $\rightarrow$  A  $\rightarrow$  B) is shared between *C* and *E*, and *C* is closer to the source than *E*. All hosts attached to router *X* are unacceptable retransmitters since packets from the source to *X* travel a completely different path than those traveling to *E*. (Note that additional factors can influence the choice of parent in AIM, LMS, and Tracer, including perceived packet loss.)

The advantage of organizing the receiver set into a tree such that every retransmitter-requester relationship is acceptable, as in AIM, is that all receivers that are downstream on the retransmitter-requester tree are also downstream on the multicast routing tree [12]. For example, in Figure 4, if *C* is the retransmitter for *D*, and *D* is the retransmitter for *E*, multicasts from *C* towards *D* (i.e., traveling from the common router *B* and away from the source) will reach all of *D*'s children, and not hosts that are not *D*'s children. We refer to a transmis-

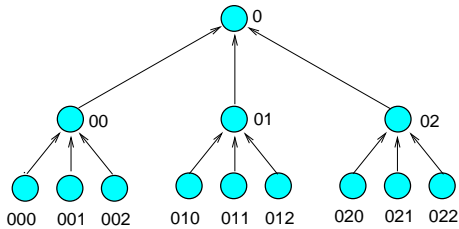


Fig. 5. The Lorax protocol [11] prefix labeling scheme.

sion to a portion of the multicast tree, e.g., starting  $B$  and away from the source (or root) and the incoming interface, as a *subcast*.

While the approach of requiring acceptability is good, AIM and LMS require several additions to the protocols used at multicast routers. In contrast, Tracer also builds a completely acceptable tree of receivers, but without requiring any changes to routers. Just like AIM and LMS, Tracer is able to identify the specific router from which subcast retransmissions should be sent.

### 2.4 Exchanging data of observed performance

A novel approach to organizing the receiver set is considered by the STORM [28] protocol, where receivers pick the best retransmitter based on observed performance, propagation delay, and buffer size. Each potential retransmitter is evaluated based on what percentage of the packets can be received at the requester before the buffer runs out. Specifically, each potential retransmitter is asked to determine the percentage of packets it has received correctly from the source within time  $t + B - d$ , where  $t$  is the average delay from the source to the requester,  $B$  is the buffer size at the requester (in time units), and  $d$  is the unicast propagation delay between requester and retransmitter.

The STORM resilient multicast paradigm works well, and includes a labeling system to avoid looping of retransmitters. However, the STORM receiver set organization is not set up properly to do efficient multicast retransmissions (via either subcasts or separate multicast groups). Multicast retransmissions do not work with STORM because measurements are performed using unicast paths, and there is nothing about measuring hop count, propagation delay, or packet-loss correlation that deterministically ensures an acceptable acknowledgment tree; nor can any of those heuristics discover the name of the router to send the subcast retransmission from.

### 2.5 Shared Acknowledgment Trees

In per-source ack-trees, all receivers know that all aggregated feedback is forwarded towards the source at the root of the tree. Building one tree per source introduces an unscalable component to an otherwise scalable scheme because each source requires construction traffic.

To resolve this unscalability, a shared ack-tree can be constructed, where all sources share the same ack-tree. Because sources may lie in arbitrary locations in

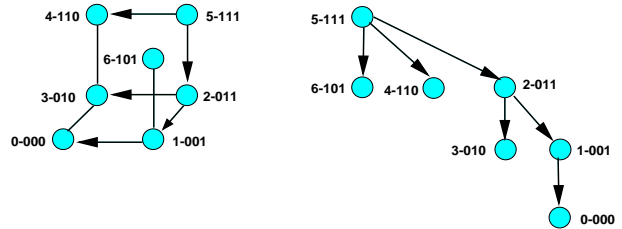


Fig. 6. The hypercube-based embedded tree scheme [15].

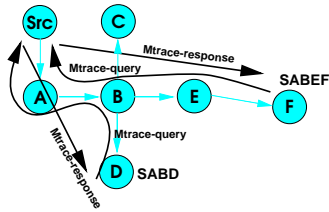
the shared tree, aggregation must be routed towards each source. The Lorax protocol [11] uses a prefix routing scheme to route acks towards arbitrary sources, and avoids routing tables at hosts. The prefix routing used by Lorax is illustrated in Figure 5. Fortunately, Lorax is compatible with Tracer. However, the acceptability condition, followed by Tracer in this paper, requires that shared ack-trees only be used when working over shared routing tree protocols, such as CBT, OCBT, BGMP, or HIP [1, 25, 10, 26]. If a shared ack tree is built for sources that all have separate, per-source routing trees, such as with DVMRP or PIM-DM, then subcast retransmissions would not follow the same path as the original transmissions, reducing the performance of the protocol.

In comparison, another shared-ack tree routing mechanism and tree-construction protocol has been proposed based on a hypercube arrangement of receivers [15]. In this protocol, the aggregation towards each source is given by a tree that is embedded in the hypercube. Each source is at the root of a distinct embedded ack-tree. A embedded tree is illustrated in Figure 6 [15]. However, this scheme follows a rule that all hosts must be packed to form a complete hypercube, regardless of topology, otherwise, the embedded ack-trees cannot be discovered. Unfortunately, packing the hosts in this fashion will not form acceptable relationships, therefore the protocol will suffer performance degradation, such as inaccurate subcasts and delayed aggregation of feedback towards sources.

## 3 Tracing

We propose a new protocol, called Tracer, which relies upon a function built into *all* IP multicast routers running the Internet Group Management Protocol (IGMP) [4]. IGMP specifies a special packet called “MTRACE” (multicast trace) that allows any host to trace its path to the source (root) of a multicast routing tree for a specified number of hops. The commonly used Unix tool `Mtrace` is also based on the MTRACE function.

Unlike hop counts, propagation delays, or exchanging performance statistics, tracing allows receivers to definitively discover if they share a common path from the source by comparing path-strings returned by MTRACE queries. Accordingly, tracing can be used by receivers to create acceptable relationships between retransmitters and requests, just as in AIM. But, unlike the mechanisms used in AIM, tracing is available without modification of routers.



**Fig. 7.** Routers *D* and *F* send an MTRACE along the reverse path to the source. The source’s router returns the recorded path-string via unicast.

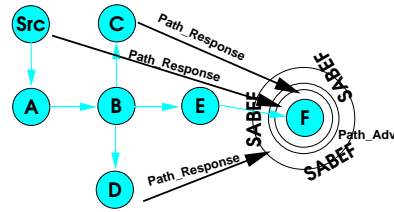
Figure 7 illustrates the MTRACE process. Because each multicast router knows the interface that leads to the source of the tree, the MTRACE query traverses each router in receiver-to-source order, recording the path in the data portion of the packet. Once the source is reached, the packet is sent back to the receiver as a standard unicast packet. Hosts that share a common path will see common network performance characteristics, as we show in Section 5.

Tracing provides requester and retransmitter with the *exact* name of the last router common to both, which enables deterministic grouping of receivers with exact packet-loss correlation, and is precisely the missing information required to do subcasting. Propagation delay and hop counts cannot provide this information, which is possibly why, although sometimes mentioned, subcasting has not been explored before now as a practical option. The introduction of subcasting to internet routers would require minimal local modification—no additional routing protocols or signaling are required. In fact, in order to perform subtree multicasting, all a router needs to do is send the packet out on all multicast interfaces appropriate for the group, except the interface leading to the source and the interface over which the subcast packet arrived on.

Tracer allows only acceptable relationships between retransmitters and requesters. Because the last common router between them is known, Tracer is able to do efficient multicast retransmissions once subcast transmissions become available in the Internet. In the absence of a subcasting interface, the groups of receivers organized by Tracer can still form separate multicast groups. Subcasting is not required to organize receivers with Tracer in order to remain compatible with the existing non-subcasting infrastructure of the Internet. However, an efficient tree construction technique based on subcasting advertisements has been suggested [14].

Tracer has four stages during which a tree of retransmitters is constructed and maintained by the host participating in a multicast group: path discovery, path advertisement, parent selection, and maintenance.

The goal of the tree building algorithm is for each receiver to pick a retransmitter that shares a common path to the source, and is closer to the source. The choice of receivers is augmented by such performance measures as propagation delay on the multicast routing tree and packet loss.



**Fig. 8.** Path advertisement by the host attached to router *F* via ERS. The source, and hosts attached to routers *C* and *D* answer in this example.

### 3.1 Path Discovery and Advertisement

Path discovery begins at each receiver by sending a unicast MTRACE message toward the source’s router. When the unicast response to the MTRACE message returns, each receiver then knows its *path-string* to the source. A path-string is a list of interfaces passed through by packets transmitted by the source to the receiver for a multicast group. Figure 7 illustrates this process.

Once the path-string is known, path advertisement begins: each receiver multicasts a PATH\_ADV message advertising its path-string and latest packet loss statistics, illustrated in Figure 8. We refer to such hosts as *advertisers*. Path advertisements also include the latest packet-loss statistics, as well as a timestamp. Advertisement is performed as an expanding ring search (ERS): the multicast is sent with a limited hop count at first, and if no response is received, the advertisement is sent again with a larger hop count. The maximum value of the ERS should not be larger than the hop count to the source (the explicit hop count can be derived from the path-string), but because of the path asymmetries between receivers communicating via source-based multicast routing trees, this should not be a hard limit. If the route taken by the advertiser’s multicast is different from the source’s multicast path, it is not a problem because only the path strings (and packet loss) are compared, not round trip times or hop counts between receivers on asynchronous paths.

Multicasting the advertisement, even as an ERS, is inefficient. However, without additional router support, there is no other way for receivers to discover each other’s existence in the multicast group. ERS by itself lacks the notion of directionality and hence simply using ERS may lead to the choice of a parent that is *downstream* from the child (See section 2.1). Tracer, on the other hand, combines ERS with path-strings to obtain a precise idea of direction. Thus, Tracer will *never* choose a parent that is farther from the source than the child is from the source.

The fundamental difference in the way Tracer uses ERS with respect to other protocols is that Tracer uses ERS to *advertise* its selection criteria, while previous protocols use ERS as an *integral part* of the selection criteria (e.g., [5, 29]). Protocols that rely on multicast advertisements as part of their selection criteria (e.g., [5, 9]) require shared multicast routing trees.

Upon reception of another host’s PATH\_ADV message, a receiver must decide whether it can respond as a willing parent, based on the acceptability definition. A

receiver should only respond to it if it can serve as an acceptable parent, i.e., if the responder is closer to the source (in terms of hop counts) and a common path exists. This can be easily decided by comparing the path strings. Hosts that do not wish to be retransmitters simply do not respond to PATH\_ADV messages, and set in their own PATH\_ADV a NON\_PARENT flag that states they do not wish to be parents. Willing parents send a PATH\_RESPONSE message unicast to the advertiser, which includes the parent’s path string, latest packet-loss statistics, and the PATH\_ADV’s timestamp.

The timestamp is used to measure potential parents during failsafe mode, described subsequently, or when delivering retransmissions in a timely manner is an issue (see Section 4.2).

Once one or several PATH\_RESPONSEs are received, an advertiser may begin parent selection. In fact, when any PATH\_ADV message is received, and if the advertiser is determined to be an acceptable *parent* (and the NON\_PARENT flag is clear), then parent selection may also begin.

Lost PATH\_RESPONSEs or PATH\_ADVs are not a problem as the tree of receivers is repaired by the maintenance stage of Tracer. Therefore, such messages do not need to be sent reliably.

The procedure in C code used to compare two path-strings appears in Figure 15. The function `is_acceptable()` returns 0 if the paths only match at the sender (and both hosts are not the sender itself), or if no part of the paths are in common. Otherwise, the number of hops between the parent and child is returned: a positive value indicates that the host advertising pathA is an acceptable parent for the host advertising pathB; a negative value indicates that the host advertising pathB is an acceptable parent for the host advertising pathA.

### 3.2 Parent selection

Parent selection is based on tracing and the packet-loss rate of a potential parent. Upon receiving a response from any potential parent, advertisers check if the potential parent is a better choice than their current retransmitter, if any. Whichever potential parent is closer on the multicast tree is the best choice, and is determined by comparing the path strings, barring packet loss, which we discuss below. An IM\_YOUR\_CHILD message is sent to the best parent confirming the relationship, and the parent stores its child’s path-string included in the IM\_YOUR\_CHILD message for use later in the protocol. A NOT\_YOUR\_CHILD message sent unicast to the old parent terminates the relationship.

Periodic IM\_YOUR\_PARENT messages must be sent from a retransmitter to all its immediate children to ensure the relationship is not broken by network partitions, or other problems. If a child does not hear from its parent after several periods, then path advertisement begins again. Initial IM\_YOUR\_CHILD messages expect an immediate IM\_YOUR\_PARENT message from the new parent, and if a response is not heard from the parent

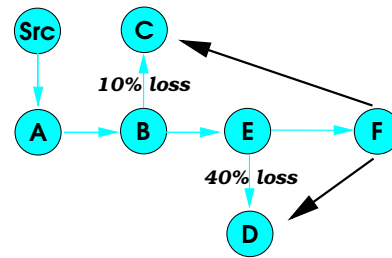


Fig. 9. Because router D is behind a lossy link, hosts at F choose retransmitters at C.

after several retransmissions, the child reverts back to path advertisement.

Tracing provides a topology-based solution, but because topology alone does not always yield the best solution, Tracer also considers the latest packet-loss measurements of a receiver. In addition, the flexibility of not electing to be a parent is also provided as it helps in load balancing. Tracer can also be extended to consider propagation delay, which is useful for delay-oriented protocols, which we describe in Section 4.2.

For example, if the closest receiver is behind a lossy or slow link that is not part of the common path, a next-closest ancestor may be chosen, as illustrated in Figure 9. A host at *F* may choose between retransmitters at *D* or *C*; notice that *D* is a closer router, but it is behind a lossy link and the host at *C* is a better choice. Given that both hosts are acceptable and that the host at *D* has a higher packet-loss rate than the host at *C*, the host at *F* may choose the host at *C* as its parent. In general, if an acceptable parent experiences more losses than its child, the parent is probably behind a lossy link that is not a part of the path that the parent and child share from the source. In the worst case, every receiver will wish to have the source be its retransmitter, or one retransmitter may become overwhelmed with too many children. To prevent this scenario, overloaded retransmitters may reject new children who have closer choices. Because the path strings of all children are known to a parent, when the load becomes too high, grandchildren can be kicked out of the group first with a NOT\_YOUR\_PARENT message. For example, the host at *C* may reject the host at *F*’s request, and force it to reply to the host at *D*.

### 3.3 Maintenance

The maintenance portion of the protocol must consider four types of changes to the multicast routing tree topology. The path of routers to the source can be altered, a retransmitter can leave the session, a new host can join the session, or a parent’s packet-loss rate may change.

To check whether the path to the source on the routing tree has changed, each receiver runs a trace periodically. However, a full trace is unnecessary; only a trace to the last router common to itself and its parent’s path to the source is required. The time-to-live (TTL) field of the MTRACE query can be restricted by the known hop count to the last common router. Any changes beyond that point will not change the status of acceptability.

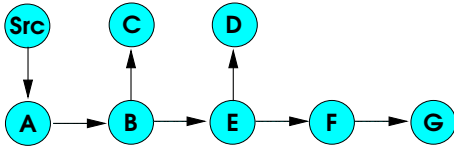


Fig. 10. A host at router  $C$  joins the session.

Tracer will adjust the tree of retransmitters with any changes to the multicast routing tree, which may induce a lot of traffic if the routing tree changes frequently. We believe this is the right approach because Tracer was designed to always pick what it sees as the best retransmitter for each requester; as the routing tree changes, some retransmitters are no longer the best choices.

If the MTRACE response does show that there has been a change in the path, the receiver must discover if its parent is still the closest acceptable node. The ERS-based path advertisement state is entered again with the distance to the parent as the starting TTL.

In the second case, where a retransmitter leaves the session, the orphaned children start the protocol over, by sending PATH\_ADVs via an ERS.

In the third case, a new receiver may join a session between an existing parent and child pair. Figure 10 illustrates this case, where a host at router  $C$  has joined a session and the host at router  $A$  is already the parent of the host at router  $F$ . After the host at  $C$  joins, it learns through an advertisement stage that the host at  $A$  is its best parent. Note the difficulty that may arise: the host at  $C$  is now the best parent for the host at  $F$ , but there is no way for the host at  $F$  or the host at  $C$  to know this. The MTRACE query from  $F$  to the source will not return a different path string, and if the PATH\_ADVs from the host at  $C$  do not reach the host at  $F$  (since the TTL needed to reach  $A$  from  $C$  during ERS is only 2), then neither  $C$  nor  $F$  will act on forming the relationship. To prevent this scenario, whenever a parent takes on a new child, it sends a copy of the new child's PATH\_ADV message to all its children (via multicast, if a separate group exists). In the case of Figure 10, the host at  $A$  would send a copy of the PATH\_ADV from the host at  $C$  to the host at  $F$ , triggering a parent selection process at the host at  $F$ . (This also repairs the situation where a host's PATH\_ADV was lost before reaching a potential child.)

In the fourth case, the parent's packet-loss rate may reach an unacceptable level. Parents advertise such information in the periodic IM\_YOUR\_PARENT messages sent to all children. If the parent's packet loss rate is much greater than the child's, the child may elect to enter the advertisement phase again to find a better parent. Note that the comparison with the child's rate is important: if both parent and child *both* have increased rates, then probably the loss is on a common path, and the parent may be already looking for a new parent. Upon confirmation of a new parent, a NOT\_YOUR\_CHILD message terminates the relationship with the old parent.

### 3.4 Failsafe operation

Because Tracer relies on the multicast path to remain stable, and assumes the correct operation of the MTRACE mechanism, an extra safeguard prevents looping in the tree if either of the two assumptions is violated. Each receiver stores a *level number*, with the source initialized as zero and receivers without parents initialized to infinity. The level of each retransmitter increases by one as the tree grows from the source. No receiver can advertise its path, or accept children in the tree until the multicast trace has been confirmed by a second trace at some subsequent time (e.g., less than a minute). If the paths remain unstable, or there is no response to an MTRACE query for a number of retries, receivers resort to a *failsafe* mode, and look for retransmitters that are closest, by measuring the sum of the multicast propagation delay from the parent to the child and the unicast propagation delay from the child to the parent (just as STORM-Tracer does in Section 4.2). To ensure loop-free operation, no requester can join a retransmitter with a lower-level designation (ties are broken by sorting IP addresses). Once a join is made, then a child takes on its parent's level number plus one. This is the only way a child may change its level. Increases in level are forwarded down the tree by a parent to its children. It is straight forward to show that loop freedom is achieved because the level numbers render a complete ordering along the multicast tree from the source.

### 3.5 Improving the Efficiency of Tracing

The MTRACE portion of IGMP is currently under revision as a separate entity [2], focusing on using MTRACE as a diagnostic tool for individual receivers of a multicast group. Accordingly, the revision aims to avoid the scenario where the source (and root) of a routing tree initiates a trace that is multicast to all receivers. Conversely, because Tracer requires *all* receivers to trace their path, it would be more efficient to initiate traces from the source that are multicast to the receivers. As the source's packet traverses the multicast tree, the path would be recorded in the data portion of the packet. Rather than each receiver periodically initiating an MTRACE query, Tracer would instead require each source to periodically trace the paths to the receiver set. Then, if a receiver detects a change in the path past the last router common to its parent, it begins the maintenance portion of the protocol. We are not suggesting that the ability for a single receiver to trace its path from the source be removed, rather just that source-based multicast tracing be added to IGMP.

While shared-tree multicast routing protocols, like OCBT [25] or BGMP [10], are not in wide use in the Internet, we expect some MTRACE mechanism will be included when they are prevalent. This section deals with issues that arise.

The MTRACE mechanism is built on the assumption that all routers know the interface that leads to the source, which is also the root of the routing tree. In a

shared tree, it is not clear through which interface an arbitrary source can be reached and the MTRACE mechanism, as it is defined now, will not work. There are two solutions to this problem. The first is to simply perform multicast traces that are initiated from each source, as proposed. The second is for each source to trace the path to the root (i.e., “core” or “rendezvous-point”) of the shared tree. Each receiver can then trace its path to any source by tracing its path to the root, and finding the common router for a particular source. The two routes are spliced at the common router to find the path between source and receiver. The first solution is obviously more efficient for multiple receivers and sources, but the second solution is useful as a diagnostic tool for individual receiver-source pairs.

## 4 Using Tracer As a Component of Larger Protocols

The set of proposed reliable multicast protocols to date can be categorized based to how each protocol handles *error control* and *congestion control*. Error-control protocols can be further divided by considering whether they provide *complete recovery* from packet loss at the expense of delay, or whether they provide low-delay, *deadline-oriented* delivery at the expense of complete recovery from packet loss. Both classes of error-control protocols can benefit from forward error correction (FEC) schemes, and many such schemes have been proposed recently (e.g., [18, 19, 9])

Tracer’s organization of the receiver set is topology-based, and therefore can be considered as a replacement component for any reliable multicast protocol that needs to organize receivers according to packet-loss correlation. The rest of this section describes how Tracer can be used as a component of representative examples of complete-recovery protocols, deadline-oriented protocols, and congestion-control protocols.

### 4.1 Grouping by Topology with Tracer

RMTP relies on *designated receivers* (DRs) to supply retransmissions to the other receivers. A receiver’s choice of DR (i.e., its parent) is based on periodic advertisements multicast by the DRs. By subtracting the number of hops left in the advertisement’s IP Time-to-Live field from the known initial value of the TTL field, receivers are able to determine how far away each DR is in hops. Receivers then chose the DR that appears to be closest according to the number of hops. Problems associated with using hop counts as the sole metric are discussed in Section 2.

Adding Tracer as a component to RMTP so as to handle auto-selection of DRs for each receiver requires no modifications to Tracer, and Tracer has no effect on the error control mechanisms designed for RMTP. The caveat is that because receivers can change DRs during the session, aggregate Acks must be used in order to ensure complete error recovery, which are Acks that

start from the bottom of the receiver tree and are aggregated towards the top<sup>1</sup>. All tree-based protocols that allow dynamic reorganization of the receiver set during the session require aggregate Acks to ensure packets are not deleted from all DRs during the reorganization [11].

Tracer could also be used as a component of the SRM [5] protocol. Rather than performing Nack-avoidance between all receivers, Tracer could be used to form separate groups of receivers, organized in a tree structure, where Nacks and Nack-avoidance timers remain local to each group. Nacks would be forwarded up the tree to the source; retransmitters that have already received the missing data correctly could subcast a repair down the routing tree.

### 4.2 Grouping by Deadlines with Tracer

Because Tracer is designed to multicast retransmissions, it is useful as a resilient multicast protocol. In other words, if an application does not require 100% of the data, such as in audio or video services where some loss is tolerable, the emphasis of a reliable protocol may be placed on getting as much of the data as possible before a deadline passes. Multicast retransmissions can lower average packet delay [22], which is desirable for the real-time streaming applications that resilient multicast protocols are designed to support.

In this situation, retransmitters should be chosen based on what percentage of the data is received correctly for a given time scale. This approach was first proposed in the STORM protocol and was called *resilient multicast*. For example, for a requester that must receive all data within 130ms, a retransmitter that receives 50% of the data within 100ms, and 65% within 300ms, is actually a better choice than another retransmitter that receives 10% of the data within 100ms and 100% within 300ms (assuming they are both within 10ms of the requester).

The STORM protocol [28] relies on unicast retransmissions, but Tracer’s approach to building the tree of retransmitters can be adapted to the resilient multicast model, with the advantage of multicast retransmissions. Just as in STORM, a STORM-Tracer combination evaluates each retransmitter based on what percentage of the packets can be received at the requester before the requester’s buffer runs out. While the original STORM protocol measures the unicast delay, STORM-Tracer retransmitters determine the percentage of packets it has received correctly from the source within time  $t + B - (M + d)$ , where  $t$  is the average delay of packet sent from the source to the requester,  $B$  is the buffer size of the requester (in time units),  $M$  is the multicast propagation delay between retransmitter and requester (measured with PATH\_ADV messages) and  $d$  is the unicast propagation delay between requester and retransmitter. The roundtrip time on the multicast tree is not needed

<sup>1</sup> RMTP does not specify aggregate Acks explicitly, but the Reliable Multicast File Transfer Protocol (RMFTP), an application based on RMTP, does already use aggregate Acks.

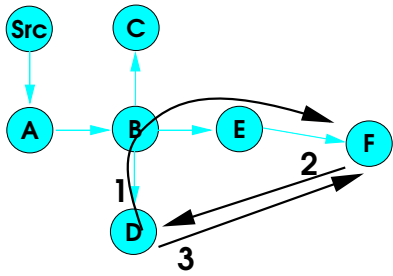


Fig. 11. Determining the multicast propagation delay between hosts

because Acks and Nacks are sent unicast to retransmitters. If subcast is to be used, then PATH\_ADVs should be subcast. The tradeoff with this approach, as opposed to the original version of STORM, is that determining the suitability of a potential retransmitter requires multicast traffic.

The measurement of  $M + d$  requires three steps, and Figure 11 illustrates the process. Each multicast PATH\_ADV includes a timestamp,  $a$  (step 1 in Figure 11). Requesters then echo  $a$  in their IM\_YOUR\_CHILD message sent unicast to the advertiser, which the advertiser receives at time  $b$  (step 2). The advertiser is then able to determine  $M + d = b - a$ . This value is returned to the requester along with the percentage of packets received correctly by time  $b - a$  (step 3). (Time spent processing at the respective hosts should be subtracted from the cumulative time.)

#### 4.3 Grouping by Congestion with Tracer

Work by De Lucia and Obraczka [3] groups receivers together with the same state of congestion. Each group has a representative that is ideally the most upstream receiver among those in the group, because such a receiver is most likely located ahead of congested links. The main problem in congestion control is for a multicast flow to share the bandwidth of a congested link with TCP flows in a fair way. TCP reacts to congestion within a roundtrip time and multicast protocols need to do the same thing. However, there may be multiple bottlenecks in a multicast flow.

In De Lucia and Obraczka’s scheme, the source builds the representative set by choosing the receiver that has not Nacked for the longest period of time because it would seem it has the least congestion. This approach avoids timer traffic and is simple to implement, but cannot guarantee that the choice of representative is optimal because Nacks and Acks may get lost during times of congestion. Handley has proposed an extension to this approach [6] based on the notion of *relays* and *subgroups*. Relays are receivers that buffer data from the source (or another relay) and re-multicast the data at a slower rate to a subgroup of receivers. Because each member of the subgroup leaves the main multicast group, the relay acts as a *representative* of the subgroup on the main multicast group, sending Acks and Nacks towards the source. Group formation is based on Nack-avoidance algorithm

specified by the SRM [5] protocol. If a receiver notices that other receivers’ Nacks are for the same or a superset of packets, the receiver reduces its Nack backoff timer. If the other receiver’s Nacks are not duplicates, or are for a subset of lost packets, then the examining receiver increases its backoff timer. The hope is that receivers with lower packet loss than sites with which they have correlated packet-loss will Nack sooner and become representatives. To find a relay, representatives start an ERS to other receivers, looking for a member with significantly lower loss rates. Once willing receivers are found, the representative starts a new multicast subgroup; the relay site is now primary source of data for all subgroup members, as they leave the source’s original group.

In comparison to De Lucia and Obraczka’s technique and Handley’s technique, Tracer’s approach to tree construction is deterministic. While the main goal of the two former schemes is for each receiver to find another receiver that is upstream from congestion, unfortunately nothing in the protocols guarantees this property. Tracer, as we have defined it in Section 3, builds relationships among receivers such that a receiver always find a parent that is upstream on the multicast routing tree. Children can compare their perceived congestion with their parents, and if it is significantly greater, break out of the main multicast group. The parent can then buffer the data from the source and relay the data to its children on a separate multicast group at a slower rate. Receivers not wanting to be a relays can remain as leaves in the tree formed by tracer by not answering multicast advertisements or by setting the NON\_PARENT flag in their PATH\_ADV messages (see Section 3).

Anytime a break out group is formed, a message advertising the new group is subcast by the relay to all its descendants. Receivers that experience congestion can switch to the new group that uses the slower rate. Tracer cannot enforce a policy that all descendants must join, but ideally this would be the case so that both streams will not be forwarded on the same paths, exacerbating congested links.

## 5 Experimental Results

To test how well topological tracing predicts packet-loss correlation, we recorded data on the Collaborative Advanced Internet Research Network (CAIRN), a research network of about 20 routers spanning the continental United States, consisting mainly of workstations running SunOS 4.1 and PCs running FreeBSD. We chose to use CAIRN rather than a few sundry hosts across the Internet because of the open access to a large number of hosts, and because we were able to collect data at almost all points in the multicast routing tree, rather than just the end points. Figure 12 shows the topology of the 13 routers in CAIRN we used to record multicast data.

To see how packet loss was correlated over the network, we sent a steady stream of data from the *ucsc* router to the 12 other routers in Figure 12. We performed six separate sessions for manageability, and we were able to collect data at the shaded nodes of the routing tree

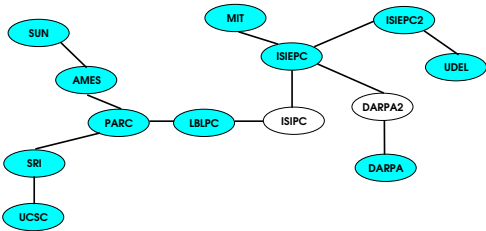


Fig. 12. Partial topology of CAIRN.

	no. of 512K packets multicast					
	4,013	3,732	3,784	13,548	19,707	9,721
sri	0.0%	0.0%	0.0%	0.0%	0.2%	0.0%
parc	21.4	24.2	29.8	—	—	—
ames	23.5	26.6	31.4	—	—	—
sun	23.5	26.7	31.4	—	72.0	72.0
lblpc	21.4	24.2	29.8	42.0	40.5	59.7
isiepc	21.4	24.2	29.8	42.0	40.5	59.7
isiepc2	21.4	24.2	29.8	42.0	40.6	59.7
mit	21.4	24.2	29.8	43.3	55.6	59.7
udel	43.5	36.6	35.1	68.0	84.7	82.8
darpa	21.4	24.2	30.7	52.8	67.6	71.5

Fig. 13. Packet loss data for all hosts for each session.

regarding which packets were received; in some sessions, data was not able to be collected at a couple hosts. Figure 13 shows the packet loss per-session for each host. In all 54,500 packets of about 512K each, or about 27 megabytes, of data were multicast from the source.

Table 14 correlates each host’s packet-loss with every host that is a direct ancestor, an indirect ancestor, an indirect descendant according to its path in the routing tree, aggregating all sessions. Each host was compared with another for only those sessions they had in common. The figure lists for each compared host, the packet-loss correlation, the ranking of the host as Tracer would have chosen retransmitters, the distance between the two hosts, and the number of routers they share in common on the path from the source.

For example, `lblpc` is compared against four hosts. Both `sri` and `parc` are direct ancestors of `lblpc`, i.e., they lie directly on the path to the source from `lblpc`. As we would expect, 100% of the packets lost at `sri` and `parc` were also lost at `lblpc`. `ames` is two hops from `lblpc` and has two hops in common on the path from the source, presented as “(2:2)”; 93% of the packets lost at `ames` were also lost at `lblpc`. `ames` is in boldface because it is the host that `lblpc` would choose as a retransmitter if no directly acceptable host was available. (Note because they are actually equidistant from the source and each other, the tie between `ames` and `lblpc` is broken alphabetically; `ames` would not choose `lblpc` as a parent.) `Sun` is an indirect descendant of `lblpc` because they share a common path from the source, but `lblpc` is closer to the source; 69% of the packets lost at `sun` were lost at `lblpc`.

As expected, the acceptability algorithm chooses the host with the highest packet-loss correlation (when ignoring directly acceptable hosts) in all cases, and the ordering of the other hosts based on acceptability exactly matches the ordering based on packet-loss correlation.

Host	Direct Ancestors	Indirect Ancestors	Indirect Descendants
<code>sri</code>			
<code>parc</code>	<code>sri</code> 100%		
<code>ames</code>	<code>sri</code> 100 <code>parc</code> 100		<code>lblpc</code> (2:2) 100% <code>isiepc</code> (4:2) 100 <code>isiepc2</code> (5:2) 100 <code>mit</code> (5:2) 100 <code>darpa</code> (6:2) 99 <code>udel</code> (6:2) 68
<code>lblpc</code>	<code>sri</code> 100 <code>parc</code> 100	1. <b><code>ames</code></b> (2:2) 93%	<code>sun</code> (3:3) 69
<code>sun</code>	<code>sri</code> 100 <code>parc</code> 100 <code>ames</code> 100	1. <b><code>lblpc</code></b> (3:2) 100	<code>isiepc</code> (5:2) 100 <code>isiepc2</code> (6:2) 100 <code>mit</code> (6:2) 97 <code>darpa</code> (7:2) 92 <code>udel</code> (7:2) 80
<code>isiepc</code>	<code>sri</code> 100 <code>parc</code> 100 <code>lblpc</code> 100	1. <b><code>ames</code></b> (4:2) 93 2. <code>sun</code> (5:2) 69	
<code>isiepc2</code>	<code>sri</code> 100 <code>parc</code> 100 <code>lblpc</code> 100 <code>isiepc</code> 100	1. <b><code>ames</code></b> (5:2) 93 2. <code>sun</code> (6:2) 69	<code>darpa</code> (3:5) 73 <code>mit</code> (2:5) 88
<code>mit</code>	<code>sri</code> 100 <code>parc</code> 100 <code>lblpc</code> 100 <code>isiepc</code> 100	1. <b><code>isiepc2</code></b> (2:5) 100 2. <code>ames</code> (5:2) 93 3. <code>sun</code> (6:2) 78	<code>udel</code> (3:5) 64 <code>darpa</code> (3:5) 82
<code>udel</code>	<code>sri</code> 100 <code>parc</code> 100 <code>lblpc</code> 100 <code>isiepc</code> 100 <code>isiepc2</code> 100	1. <b><code>mit</code></b> (3:5) 97 2. <code>darpa</code> (4:5) 96 3. <code>ames</code> (6:2) 97 4. <code>sun</code> (7:2) 95	
<code>darpa</code>	<code>sri</code> 100 <code>parc</code> 100 <code>lblpc</code> 100 <code>isiepc</code> 100	1. <b><code>isiepc2</code></b> (3:5) 100 2. <code>mit</code> (3:5) 100 3. <code>ames</code> (6:2) 93 4. <code>sun</code> (7:2) 88	<code>udel</code> (4:5) 76

Fig. 14. Observed packet-loss correlation on CAIRN. Acceptability ties are broken alphabetically.

Therefore, we conclude that tracing is a good predictor of packet-loss correlation.

The goal of choosing retransmitters based on acceptability is not to find a retransmitter with the lowest packet loss. Rather, the goal is to find the retransmitter with the highest packet-loss correlation, and that is the closest to the source. When this is the case, the parent will probably lose a packet that the child has lost.

## 6 Conclusion

Tracer addresses both error control and congestion control aspects of a multicast protocol, and is applicable to reliable multicast protocols providing complete error recovery, as well as reliable multicast protocols providing deadline-oriented recovery. While previous reliable multicast protocols attempt to discover the topology of the underlying multicast tree by measuring hop counts or propagation delay, Tracer provides a mechanism that uses currently available router functions to record the exact multicast route from the source to each receiver. Because the path is known, Tracer is able to select a retransmitter for each receiver such that packet-loss correlation, is maintained between the hosts. Tracer uses

the first technique that organizes the receiver set and maintains packet-loss correlation without changes to the routers. Therefore, Nacks for missing data are sent to retransmitters who are closer to the source, and therefore have a better chance of having the data and will receive the retransmission faster. Furthermore, Tracer is the first scheme to identify the last common node between retransmitters and requester so that subcast transmissions can become a practical solution, with only small local additions to routers. Accurately organizing receivers according to packet-loss correlation ensures that retransmissions can be multicast because only nodes expecting the retransmission lie along the multicast subtree. In addition, Tracer provides elegant mechanisms to isolate congested subtrees and elect representatives, or relays for efficient congestion control in reliable multicast protocols. Tracer could also be used to organize receivers in hierarchical distributed web cache protocols or USENET distribution protocols.

We have identified that extending IGMP with MTRACE packets multicast from sources to receivers is a desirable feature that makes Tracer much more scalable. We believe that such a feature can be used by other multicast protocols.

## References

1. A. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM'93*, pages 85–95, October 1993.
2. S. Casner and B. Fenner. A "traceroute" facility for IP multicasting. Technical Report draft-ietf-idmr-traceroute-ipm-02.txt, IETF draft, December 1997.
3. D. De Lucia and K. Obraczka. Multicast feedback suppression. In *Proc. IEEE INFOCOM'97*, pages 463–70. IEEE, 1997.
4. W. Fenner. Internet group management protocol, version 2. Internet-Draft, February 1996.
5. S. Floyd et al. A reliable multicast framework for light-weight sessions and application level framing. In *Proc. ACM SIGCOMM'95*, pages 342–356, August 1995.
6. M. Handley. A congestion control architecture for bulk data transfer. Presentation at IRTF meeting, September 1997. Cannes, France.
7. M. Hofmann. A generic concept for large-scale multicast. In *International Zurich Seminar on Digital Communication (IZS'96)*, Springer Verlag, pages 95–106, February 1996.
8. V. Jacobson. Congestion avoidance and control. In *SIGCOMM'88*, pages 314–328. ACM, 1988.
9. R. Kermode. Scoped hybrid automatic repeat request with forward error correction (SHARQFEC). In *SIGCOMM'98*, September 1998.
10. K. Kumar et al. The MASC/BGMP architecture for inter-domain multicast routing. In *Proc. ACM SIGCOMM 98*, September 1998.
11. B.N. Levine, D. Lavo, and J.J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proc. Fourth ACM Multimedia Conference (Multimedia'96)*, pages 365–376, November 1996.
12. B.N. Levine and J.J. Garcia-Luna-Aceves. Improving internet multicast with routing labels. In *Proc. IEEE International Conference on Network Protocols*, pages 241–50, October 1997.
13. B.N. Levine and J.J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems Journal (ACM/Springer)*, 6(5), August 1998.
14. D. Li and D. R. Cheriton. OTERS (On-Tree Efficient Recovery using Subcasting): A reliable multicast protocol. In *Proc. IEEE International Conference on Network Protocols (ICNP'98)*, pages 237–245, October 1998.
15. J. Liebeherr and B.S. Sethi. A scalable control topology for multicast communications. In *Proc. IEEE INFOCOMM'98*, pages 1197–204, San Francisco, CA, March 1998.
16. D.L. Mills. Internet time synchronization: The network time protocol. Network Working Group Request for Comments: 1129, October 1989.
17. T. Montgomery. A loss tolerant rate controller for reliable multicast. Technical Report NASA-IVV-97-011, NASA, August 1997.
18. J. Nonnenmacher, E.W. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proc. ACM SIGCOMM'97*, pages 289–300, September 1997.
19. J. Nonnenmacher, M. Lacher, M. Jung, E.W. Biersack, G. Carle. How bad is reliable multicast without local recovery? In *Proc. IEEE INFOCOM'98*, March 1998.
20. C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proc. IEEE INFOCOM'98*, pages 1118–1196, 1998.
21. S. Paul, K. Sabnani, J. Lin, S. Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.
22. S. Pejhan, M. Schwartz, and D. Anastassiou. Error control using retransmission schemes in multicast transport. *IEEE/ACM Transactions on Networking*, 4(3):413–27, June 1996.
23. S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Performance Evaluation Review*, volume 22, pages 221–230, May 1994.
24. P. Sharma, D. Estrin, S. Floyd, and L. Zhang. Scalable session messages in SRM using self-configuration. Technical Report 98-670, USC, February 1998.
25. C. Shields and J.J. Garcia-Luna-Aceves. The ordered core based tree protocol. In *Proc. IEEE INFOCOM'97*, pages 884–91, April 1997.
26. C. Shields and J.J. Garcia-Luna-Aceves. Hierarchical multicast routing. In *Proc. Seventeenth Annual ACM SIGACT-SIGOPS Symposium on principles of distributed computing (PODC 98)*, Puerto Vallarta, Mexico, June 28–July 2 1998.
27. T. Speakman et al. Pretty good multicast. Technical report, internet draft, January 1998.
28. X. Rex Xu, A.C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *NOSSDAV*, pages 183–94, 1996.
29. R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proc. ACM Multimedia*, pages 333–44, November 1995.

