# Fault Isolation in Multicast Trees [*]

Anoop Reddy Ramesh Govindan Deborah Estrin

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292, USA
{areddy,govindan,estrin}@isi.edu

## ABSTRACT

Fault isolation has received little attention in the Internet research literature. We take a step towards addressing this deficiency, exploring robust and scalable techniques by which multicast receivers can (in some cases, approximately) locate the on-tree router responsible for a route change, or the link responsible for significant packet loss. A common property of our techniques is that receivers with overlapped paths coordinate to share the responsibility of monitoring paths to the source. Our techniques assume no additional path monitoring capability other than that provided by *multicast traceroute* (mtrace).

We explore the scaling characteristics of two classes of mechanisms: those that assume some kind of router assist for selectively multicasting the responses to mtrace requests, and those that do not. In the former category fall schemes that use *subcast* or *directed multicast*. In the latter are schemes that use *scoping*, or *limited multicasts*. The latter two approaches can be deployed in today's multicast infrastructure. We find that while one deployable alternative has somewhat acceptable performance, schemes that leverage router assist have very desirable scaling characteristics.

## 1. INTRODUCTION

While much recent research has focused on IP multicast routing protocols and the design of multicast applications, scalable solutions for managing networks with deployed multicast has not received much attention. In this paper, we take a step in this direction, by considering the problem of *fault isolation* in the context of large multicast distribution trees. To more fully explore the design space, we focus on *single-source* trees. Such trees are fairly representative of the kinds of applications network-layer multicast is very well suited for—network broadcasts, streaming media type applications *etc.*

[*]This work was supported by the Defense Advanced Research Projects Agency under grant DABT63-98-1-0007. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*SIGCOMM '00,* Stockholm, Sweden.
Copyright 2000 ACM 1-58113-224-7/00/0008…$5.00.

The problem of fault isolation is that of *locating* (perhaps approximately) that on-tree router or link which is the origin of a *fault*. In this paper, we use the term fault in two very specific ways. First, an on-tree link is faulty if it is the cause of significant packet loss. Our mechanisms do not rely on the term "significant packet loss" being precisely quantified. Rather, multicast receivers determine which link or links contribute most to perceived packet loss. Second, an on-tree router is faulty if it is the origin of a routing change that results in a change to at least one receiver's path to the source. This definition includes the scenario in which a receiver loses connectivity to the root.

Two considerations enable us to find feasible solutions to this problem of fault isolation. First, we believe it is acceptable for the fault isolation mechanisms to localize a fault to within some topological neighborhood of the actual fault location. Simultaneously achieving perfect *spatial fidelity* of isolation, scale and robustness seems intractable. For a similar reason, we believe it is acceptable for the mechanisms to achieve imperfect *temporal fidelity*—*i.e.,* altogether miss extremely short-lived faults.

Finally, a desirable solution for multicast fault isolation must also be *robust* to receiver joins and leaves, or network dynamics. In particular, the joining or leaving of a receiver must not render the fault isolation capability inoperable at other receivers. When a network partition results in some receivers being unreachable from the source, the remaining receivers must still be able to isolate faults.

SNMP-based monitoring systems [13, 10, 24, 1, 3] provide a capability that is *complementary* to fault isolation. That is, once a fault has been located, SNMP-based monitoring systems can be used to infer the *causes* of such faults (*e.g.,* failed routers, chronically underprovisioned links). For this reason, inferring the cause of a fault is an explicit non-goal of our fault isolation mechanisms. However, SNMP-based monitoring systems cannot, we believe, be used for fault isolation. Monitoring *individual* routers may not be sufficient for correlating an application-perceived behavior with router activity. For example, given that a route change has occurred at a router, the router cannot infer the specific receiver or receivers affected by a route change. The multicast routing state in a router does not, for scaling reasons, contain the identities of downstream receivers. Thus, collecting information from *inside* the network may not be sufficient for fault isolation.

In our approach, receivers at the *edge* of the network periodically probe the path to the source. They each maintain some *history* of the results of these probes. Each receiver also *coordinates*—exchanges probe information with—some set of other receivers. Once a fault is detected, an affected receiver can isolate the fault using the probe *history*, perhaps after obtaining more detailed information from some other receivers.

The probing primitive that forms the basis of this approach is *multicast traceroute* [7]. Using multicast traceroute, a receiver on a multicast distribution tree may trace its current path to the source. In its simplest form, this is initiated by the receiver host sending an `mtrace request` message to its first hop router on the multicast distribution tree (Figure 1(a)). That router performs two actions. First, it appends its own identity (*i.e.,* its IP address) to the request message. It also appends a count of the total number of multicast packets received on this tree. This count helps determine how many packets were dropped by this router. Second, the router forwards the request to the previous hop towards the source (the identity of the previous hop is determined from routing tables). This router, and each successive router, repeats these actions. Finally, the router attached to the source returns an *mtrace response* to the destination specified in the mtrace query; this response message contains the information accumulated at each hop to the receiver.

Using multicast traceroute (or *mtrace* for short), a receiver can determine both its path to the source (or part thereof), and the number of losses on individual links in that path. Mtrace's loss statistics are approximate, and sometimes erroneous [7]. However, for fault isolation, this *qualitative* loss indication is usually sufficient. Is mtrace alone also sufficient for locating the origin of a routing change? No. If a receiver were to conduct an mtrace to the root before and after a routing change, it may not be able to infer the fault location from the two mtraces alone (Figure 1(c)). Furthermore, *a solution where every receiver mtraces to the root does not scale well to large trees* (Figure 1(b)). This scaling requirement motivates the study of the various mechanisms considered in this paper.

In this paper, we discuss a class of solutions that leverages the following simple observation. Consider two receivers $A$ and $B$ in a multicast tree that share a common ancestor[1]. It suffices for one of them, say $A$ to mtrace to the root, and the other $B$ to mtrace up to the common ancestor. This observation allows us to *scale* the periodic monitoring of multicast tree paths. If $A$ and $B$ share the results of their mtraces, they can each learn from the other about tree changes or lossy links. As we describe later, a receiver can approximately isolate a failed router by determining the paths, before and after a fault, of all other receivers with whom it has a common ancestor.

[1]In general, receivers can share multiple ancestors. In this paper, common ancestor refers only to the shared ancestor that is nearest to the receivers.

Receivers determine peers with whom they share a common ancestor by exchanging mtrace results with other receivers. In our example above, $A$ would *subcast* (i.e. subtree multicast [17, 4]) its response. Upon receiving this, $B$ would infer that it shares an ancestor with $A$, and would limit its mtrace to that shared ancestor. In turn, $B$ would subcast its response from the shared ancestor. Subcast is not, however, the only possibility; other possibilities include directed multicast [20], scoped multicast, or multicasted mtrace responses from a limited subset of receivers.

In this paper, we describe our approaches in some detail (Section 2). We then attempt to compare the performance of these schemes. The performance metrics of interest (the probing overhead and the fidelity of fault isolation) are dependent on the characteristics of the multicast tree. We analytically evaluate these metrics for a class of regular trees (Section 3.1). We then simulate the performance of these schemes over a wider, parameterizable, class of randomly generated trees (Section 3.2).

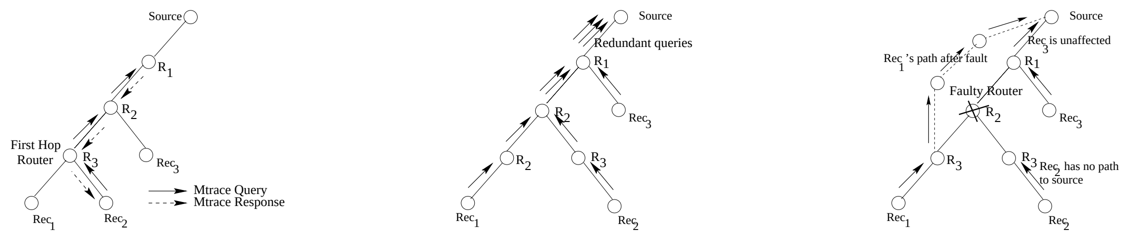
We find that the scoped-multicast based fault isolation has good average performance. However, schemes that employ router assist have very desirable (logarithmic) overhead scaling, and high fault isolation fidelity. This is particularly true of directed multicast. Our results confirm the need for router assist in a deployed multicast infrastructure, and demonstrate the existence of protocols other than reliable multicast that can utilize such assist.

## 2. MULTICAST FAULT ISOLATION

In this section, we show how multicast traceroute can be used for scalable and robust fault isolation in multicast trees. For clarity, we use the term *session watcher* to denote the software entity that actually initiates the multicast traceroute, keeps probe history, and coordinates with other session watchers. A session watcher may be colocated with a receiver. More generally, there may be one session watcher in a multi-access LAN, performing fault isolation on behalf of all receivers on the LAN.

There exists a naive technique for monitoring multicast distribution trees that uses multicast traceroute. Each session watcher periodically traces its path to the source and maintains a history of these traces. Using this information alone, a session watcher (denoted by $W_a$, say) can identify the link responsible for significant loss. Locating the router responsible for a route change is harder: $W_a$ needs to query neighboring session watchers to determine if, before the route change, they happened to share a part of the path of the source. If $W_b$'s path to the source was unchanged after the route change observed at $W_a$, then the location of the fault must be downstream of ancestor common to $W_a$ and $W_b$. Otherwise, the fault location must be upstream of this ancestor. $W_a$ can localize the fault by this process of elimination. This is illustrated in Figure 1(c).

This naive technique has some interesting properties. Other session watchers are not affected by the failure of a session watcher. The temporal fidelity of this technique is governed by the rate at which session watchers send mtrace requests; this technique will not be able to isolate faults whose duration is less than the interval between mtrace requests. Fur-



(a) Multicast Traceroute query from $Rec_2$ is forwarded hop by hop upto router $R_1$. $R_1$ constructs the response and sends it to the specified destination, $Rec_2$.

(b) Naive Approach: Redundant monitoring results when all receivers probe upto the source.

(c) $Rec_1$ needs path information before and after the fault from $Rec_2$ and $Rec_3$ and itself in order to isolate the fault at $R_2$.

**Figure 1: *Multicast Traceroute***

thermore, this technique scales better to large groups than a centralized collection of traceroutes (using, for example, third-party [7] multicast traceroutes).

Nevertheless, this technique still exhibits undesirable scaling behavior. Closer to the root of the distribution tree, the overhead of mtrace requests can be significant, particularly for very large multicast groups. Furthermore, to isolate a routing change, a session watcher may need to query potentially every other session watcher in the group.

## 2.1 Subcast-Based Fault Isolation

The first fault isolation technique we describe, subcast based fault isolation, relies on two capabilities. First, it relies on the ability of a session watcher $W_a$ to specify a *hop limit* $h$ on an mtrace request. This feature already exists in mtrace [7]. A hop-limited request travels on the reverse path from receiver to source, terminating at a router $R$ which is $h$ hops away from the receiver. That router generates the corresponding response. We say that $R$ is $W_a$'s *turnaround router*. When the hop limit on an mtrace response is $\infty$, the turnaround router is the root (the router directly connected to the source). Second, this technique uses *subcast*, or subtree multicast [17]. This form of router support allows a sender to specify that a packet be multicast to the subtree rooted at a specified router. In particular, we assume that we can specify that the corresponding mtrace response be subcast at the turnaround router. Subcast is illustrated in Figure 2(b).

We should emphasize that subcast is not yet widely deployed in the Internet's multicast infrastructure. However, because it has wide applicability for reliable multicast, there is some reason to believe that it will be implemented in future multicast-capable internets. Furthermore, as we show later, this subcast based scheme helps us calibrate the performance tradeoffs of other immediately deployable designs.

### Overview

Our subcast-based session watcher coordination leverages the following observation. If the path from two session watchers to the source *overlaps*, it suffices for one of them to monitor the overlapped segment of the path. Thus, one session watcher, say $W_a$ can monitor the entire path to the source. The other, $W_b$ can terminate its multicast traceroute—using the hop-limit field in the mtrace request—at the common ancestor between $W_a$ and $W_b$ (Figure 2(a)). An earlier version of this algorithm appears in [22].

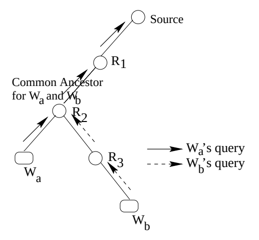
How does $W_b$ determine (1) that $W_a$ is tracing its path all the way to the source, and (2) the identity of the common ancestor? When $W_b$ starts up, it mtraces its path once to the source[2]. Further, $W_a$'s response is sent to the entire tree and is therefore heard by $W_b$. More generally, if $W_a$ itself terminates its mtrace at some non-root node, it subcasts its response at that router. Using this subcast response, $W_b$ can determine the answers to both these questions. Having determined its common ancestor, $W_b$ then subcasts its mtrace response down the subtree rooted at the common ancestor. Could $W_a$ (instead of $W_b$) have terminated its mtrace request at the common ancestor, allowing the latter to send a multicast traceroute to the source? In our scheme, we use a simple metric (the number of hops to the common ancestor) to determine which of $W_a$ or $W_b$ traces all the way to the source.

Finally, what happens if $W_a$ fails? When $W_b$ fails to hear a small number of subcast mtrace responses from $W_a$, it extends its mtrace request all the way to the source. (If several session watchers terminated their mtraces at that common ancestor, then—as we describe below—only one of them mtraces beyond the ancestor.) This is a conservative action. For example, $W_a$ may not have failed; rather, several of $W_a$'s subcasts may have been dropped. In that case, $W_b$'s action only results in redundant monitoring; for a short time, $W_b$ and $W_a$'s mtraced paths may overlap. Once $W_b$ hears $W_a$ subcasts again, it *backs off* its mtrace requests to the common ancestor.

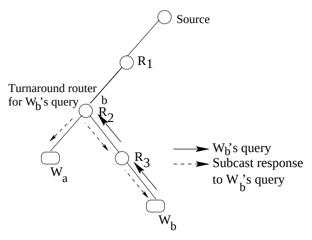
### Protocol Description

In the following paragraphs, we describe subcast based coordination of path monitoring. We do this by describing the sequence of actions executed at each session watcher $W_a$. First, we describe some notation:

[2]This is a simplified description. On startup, a session watcher actually "hunts" its path to find an ancestor. This behavior is described later in this section.



(a) In this figure, $W_b$ only monitors upto its common ancestor with $W_a$. Every link is monitored by only one session watcher.


Source
R1
Turnaround router for Wb's query
b
R2
R3
Wa
Wb
Wb's query
Subcast response to Wb's query

(b) In this example we illustrate subcast. $W_b$'s query terminates at its turnaround router, $R_2$. The response is subcasted at $R_2$ and is received by $W_a$ and $W_b$.

**Figure 2: Subcast-based monitoring coordination**

- The term *current hop limit* denotes the hop limit that $W_a$ will use on its next mtrace request.
- $W_a$ sets a periodic *probe timer*. The duration of this timer is some fixed protocol constant $T$, randomized to avoid synchronization effects. This timer defines the periodicity of sending mtrace requests.
- $W_a$ maintains a *current turnaround router*. The current turnaround router is the furthest known ancestor to which $W_a$ is closer than all other session watchers who share that ancestor. In addition, $W_a$ maintains an *ancestor list*. This list, sorted by the number of hops to an ancestor, contains all ancestors that are turnaround routers for any other session watcher $W_b$.
- $W_a$ also sets a *backoff timer*. The duration of this timer is a small integer multiple of $T$. As shown below, this timer is used to recover from failures.
- Finally, $W_a$ also maintains a *history buffer* which is a time-stamped list of mtrace responses it has seen.

*Startup:* Upon startup, $W_a$ initializes its current hop limit to 1. It empties the ancestor list and the current turnaround router, and starts the probe timer. The basic idea here is that a session watcher "hunts" its path to the source slowly until it finds another session watcher with a common ancestor. This behavior is necessary in order to prevent all session watchers from simultaneously sending mtraces to the root after, for example, a major tree reconfiguration.

*Probe timer expiration:* When a probe timer fires, $W_a$ sends an mtrace request. The hop limit on this request is $W_a$'s current hop limit. $W_a$ also ensures that the response is subcast from the router that is at a distance indicated by the current hop limit. If $W_a$ has no current turnaround router, it doubles its current hop limit (unless its current hop limit is already sufficient to reach the source). This last step exponentially increases the hop limit until a suitable turnaround router is found.

*Receiving own mtrace response:* When $W_a$ receives a response to an mtrace request that it originated, it stores the response in its history buffer. If the current response has a different list of routers than the response elicited by $W_a$'s previous request, $W_a$ sets its current hop limit to 1 and clears out its ancestor list and its current turnaround router. In this way, it resumes its hunting behavior to find a suitable turnaround router. Obviously, during this hunting, the list of routers in the response is different from the response in its history buffer. Therefore, during hunting, the current hop limit is not reset to 1 although the list of routers in the response and the history buffer are different.

*Receiving another watcher's response:* When $W_a$ receives a response to an mtrace request sent by $W_b$:
1. $W_a$ tries to determine the ancestor $r_{ab}$ that is common to both $W_a$ and $W_b$. It can determine this using the received response and its history buffer. From this information, $W_a$ can also determine its own distance (in terms of hops) to $r_{ab}$ (call this $d^a_{ab}$) and $W_b$'s distance to $r_{ab}$ (call this $d^b_{ab}$).
2. If $r_{ab}$ is not in the ancestor list, $W_a$ inserts $r_{ab}$ and its associated distances into the list.
3. $W_a$ sets its current hop limit to $d^a_{ax}$ such that $r_{ax}$ is the nearest ancestor for which $d^a_{ax} > d^x_{ax}$ (ties are broken by lower IP address). It also sets the current turnaround ancestor to be $r_{ax}$, if such a router exists.
Intuitively, this step might cause $W_a$ to *backoff* to the current turnaround router.

*Backoff timer expiration:* Suppose that, when the backoff timer expires, $W_a$'s current turnaround router is $r_{ab}$. Clearly, the backoff timer must have expired because successive subcasts from $W_b$ were not received. In this case, $W_a$ deletes $r_{ab}$ from the ancestor list, then recomputes its current hop limit and current turnaround router according to Step 3 in the previous paragraph. $W_a$ restarts its backoff timer. This step essentially renders the mechanism robust to receiver leaves and network partitions. It also ensures that, of all the session watchers for whom $r_{ab}$ is an ancestor, at most one session watcher mtraces beyond that router when $W_b$ is perceived to have failed.

Figure 2(a) shows the operation of the subcast based protocol in the steady state. The following are some of the interesting properties of this protocol. First, in the steady state, at most one session watcher monitors a link in the multicast distribution tree. Second, a session watcher $W_a$ knows all other watchers that have turnaround routers on $W_a$'s path to the source. This information can be obtained from the subcasts that $W_a$ receives. This property is crucial for fault isolation and is described below. Third, this protocol automatically adapts to receiver joins and leaves. When a receiver joins the group, the corresponding session watcher initiates mtraces using the hunting behavior described above. Eventually, it backs off to the appropriate ancestor. When a receiver leaves, the corresponding session watcher stops sending mtraces. Eventually, a backoff timer expires at some watcher, which then appropriately adjusts its hop limit to enable complete tree monitoring. Fourth, the protocol behaves conservatively in the face of packet loss, or watcher failure. For example, when mtrace responses are frequently lost, a session watcher may terminate its mtrace requests further away than it otherwise might have. This can result in redundant link monitoring. Watcher failure is analogous to receiver leaves. Failure of a single watcher affects only the receivers located on the same LAN. Finally, the state maintained at a session watcher is proportional to the number of messages seen at that watcher within one monitoring interval. This state is highly compressible when there are no changes in the receivers' paths or in losses observed at routers. However, in the worst case the amount of state that needs to be maintained is proportional to the maximum monitoring overhead.

### *Fault Isolation*

We have, thus far, described a technique for scalably monitoring a multicast distribution tree. This technique has one key property: from its history buffer, a session watcher $W_a$ can compute its ancestor list. For each ancestor in the list, it can also compute the single session watcher whose mtrace extended beyond that ancestor. We use the term $\mathcal{W}_a$ to denote this set of session watchers. This property is very useful for isolating a fault. We now describe how a session watcher may (1) locate a lossy link in its path to the source and (2) approximately locate the origin of an observed route change.

To isolate a lossy link, $W_a$ need only refer to its history buffer. From the most recent mtrace responses received from each session watcher in $\mathcal{W}_a$, $W_a$ can determine the mtrace response that it would have received if it had sent an mtrace to the source. This information suffices to infer lossy links on the path from $W_a$ to the source. Thus, our session watcher coordination protocol isolates lossy links without loss of spatial fidelity.

Suppose $W_a$ notices a route change in an mtrace response. Suppose further that $\mathcal{W}'_a$ is the value of $\mathcal{W}_a$ before a route change. Then, the router that originated the route change must lie between the nearest ancestor $r_{ax}$ in the ancestor list for which $W_x$'s path to the source has not been affected by the route change, and the ancestor closer to $W_a$ than $r_{ax}$ (Figures 3(a) and 3(b)). (Recall that $r_{ax}$ is the common ancestor for $W_a$ and $W_x$—here we refer to the ancestor relationship before the route change). Thus, in order to isolate a route change, $W_a$ will need to query, in the worst case, all the watchers in $\mathcal{W}'_a$.

The subcast based scheme (and indeed other schemes described in subsequent subsections) can only localize a fault to between branch points in a multicast tree. There may be many routers between two branch points. Thus, the *spatial fidelity of locating a route change depends upon the distribution tree.* However, as discussed before, $W_a$ at least knows the identities of these routers and can use other techniques (*e.g.,* SNMP) to more precisely locate the fault.

### *Metrics*

Before we proceed to describe other approaches, we must clearly define what determines the performance of these fault isolation schemes.

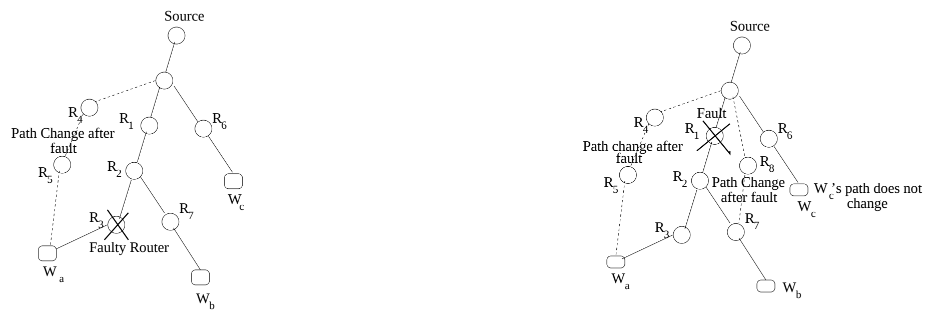
Our first metric describes the overhead of the mtrace path probing. We define the overhead on a link $l$, $O_l$, to be the number of messages (requests and responses) seen on $l$ in a time window $T$, the probe timer interval. Intuitively, $O_l$ measures the number of messages that would be seen on link $l$, in the absence of losses, if every session watcher sent an mtrace (with its current hop limit in the subcast-based scheme, of course) exactly once. Given that probe timer expirations at session watchers are not synchronized, we measure $O_l$ as a long-term average number of messages seen on $l$ within an interval of $T$.

For our performance analyses (Section 3), we define two overhead measures: the *maximum overhead*, and the *average overhead*, defined across all links. These overhead measures are clearly dependent on the characteristics of the multicast trees; the number of session watchers, and the distribution of router fanout. The overhead is also dependent on link losses; in the subcast based scheme, if some responses are lost, receivers may redundantly monitor links in the distribution tree, resulting in increased overhead.

In this paper, we make two simplifying assumptions in evaluating overhead. First, we count only individual messages, and ignore message size. To a first approximation, we expect mtrace requests and responses to be relatively small (a few tens to a few hundreds of bytes). Furthermore, the costs of processing mtrace requests are, again to a first approximation, per packet not per byte. Second, we model multicast tree links as point-to-point links. In some cases, several downstream routers may be attached to the same LAN as their upstream neighbor. In these cases, $O_{ave}$ will be different from—and could be higher or lower than—the true average overhead. For the example in Figure 2(a), the average overhead is 3.33 and the maximum overhead is 4.

Our second metric defines the accuracy of fault isolation. In the subcast scheme, depending on tree characteristics, a session watcher $W$ may be able to isolate a fault at router $R$ to within $n$ routers on the same tree branch as $R$. In this case, we say that the fault isolation *error* for $R$ with respect to $W$ is $n - 1$. We define the *fault isolation error of a router $R$* as the maximum fault isolation error with respect to all session watchers affected by the fault at $R$. We then define the *average error* over all routers $R$ on the tree. Intuitively, the average error gives the *expected* fault isolation inaccuracy if each on-tree router were equally likely to fail (or not have a route to the source). We emphasize that this metric *is only relevant* for isolating a failed router or the origin of a route change. Receivers can always isolate lossy links with perfect fidelity, modulo inaccuracies in the loss data provided by mtrace [7].

Clearly, for the subcast scheme, both the overhead and the error are dependent on tree characteristics. However, as we



(a) The paths for $W_b$ and $W_c$ do not change after the fault at $R_3$. $W_a$ isolates the fault at $R_3$

(b) $W_a$ isolates the faulty routers to the set $R_1$, $R_2$ assuming only a single fault has occurred in the system. If multiple faults were assumed, the faulty router set isolated by $W_a$ would be $R_1$, $R_2$ and $R_3$.

**Figure 3: Fault Isolation Scenarios**

show later, the fault isolation error for subcast is lower than the other schemes we now describe.

## 2.2 Fault Isolation Using Directed Multicast

Directed multicast [20] is a proposed router assist mechanism for reliable multicast applications. Unlike subcast, where a packet is multicast down all branches of the subtree rooted at a specific node, directed multicast allows receivers to multicast the packet along a specific branch. A directed multicast message sent by a turnaround router is received by all members of the subtree downstream of the link by which the corresponding query arrived at the turnaround router. This is illustrated in Figure 4(a).

Directed multicast can be used as a drop-in replacement for subcast in the previous algorithm. Specifically, when the probe timer expires, a session watcher indicates that the mtrace response be directed-multicast at its turnaround router. No other change need be made to subcast-based coordination of path monitoring.

Directed multicast based fault isolation has very different characteristics compared to subcast based fault isolation. First, the overhead of this scheme can be significantly less than subcast based fault isolation. Second, the fault isolation error can be higher. In some cases, the maximum error can equal the distance from a receiver to the source. However, as we show later, the average error for the directed multicast based scheme is quite acceptable over a wide variety of multicast trees. Finally, in the directed multicast based scheme, the fault isolation error of a router $R$ with respect to a session watcher $W$ is *asymmetric*; *i.e.,* two session watchers might isolate the same fault with different error values.

## 2.3 Fault Isolation Using Scoped Multicast

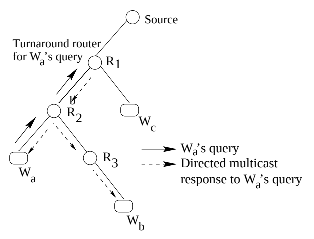
The schemes presented in the previous sections rely on router support for directionally forwarding mtrace responses. An alternative approach, which doesn't require router support, uses *TTL-based scoping.* Essentially, this approach leverages the feature of multicast traceroute to both multicast the response, and to hop-limit the response. This approach, then, is almost identical to the subcast based approach—rather than subcast the response, a session watcher indicates that its turnaround router should multicast the response with a large enough *response hop limit.* This hop limit should be sufficient to reach all session watchers that have $R$ as an ancestor (Figure 4(b)).

How does a session watcher $W_a$ compute its response hop limit? It initially sets the response hop limit to the current hop limit (Section 2.1). Suppose that $W_a$'s turnaround router is $R$. Suppose further it sees a response from some other session watcher whose turnaround router is also $R$, and the length of the path in the response is $L$. If $W_a$'s response hop limit is smaller than $L$, it sets it to $L$. In this manner, all session watchers that share $R$ as an ancestor rapidly converge to the correct response hop limit. Various details of this scheme such as current hop limit determination and the response hop limit decrease algorithms are not described here due to space constraints.

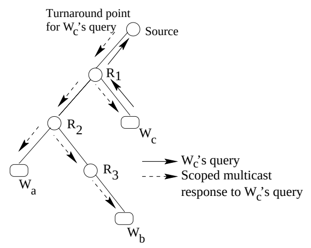
Clearly, the scoped multicast based scheme has greater overhead than subcast. Its responses not only traverse the subtree rooted at the turnaround router, they also traverse up the multicast tree. However, the fault isolation error of scoped multicast is identical to subcast based schemes. The scoped multicast scheme has one important drawback—TTL-based scoping does not work with multicast routing protocols that construct unidirectional shared trees, such as PIM-SM version 2 [5]. PIM-SM is deployed in parts of the Internet infrastructure today. More recent versions of this protocol [6, 9] are likely to support bidirectional shared trees. Such trees do not invalidate TTL-scoping semantics.

## 2.4 Fault Isolation Based on Limited Multicasts

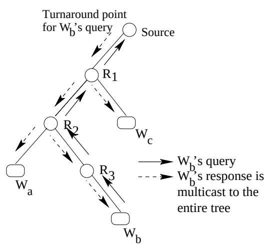
In this section, we consider a solution in which only a relatively small number of session watchers multicast, to the entire group, their mtrace response (the mtrace protocol already allows a receiver to multicast mtrace responses to the group). Every other session watcher mtraces along part of its path to the source; the response is returned to that session watcher. This solution is attractive because it would, if


Source
Turnaround router for $W_a$'s query
$R_1$
$R_2$
$R_3$
$W_a$
$W_b$
$W_c$
$W_a$'s query
Directed multicast response to $W_a$'s query

(a) Here we illustrate the *directed multicast* based scheme. The response to $W_a$'s query is only heard by $W_a$ and $W_b$ as opposed to the subcast based scheme where even $W_c$ would hear $W_a$'s response


Turnaround point for $W_c$'s query
Source
$R_1$
$R_2$
$R_3$
$W_a$
$W_b$
$W_c$
$W_c$'s query
Scoped multicast response to $W_c$'s query

(b) In the *scoped multicast* based scheme the scope of the response to $W_c$'s query is 4 *i.e.,* it reaches both $W_a$ and $W_b$.


Turnaround point for $W_b$'s query
Source
$R_1$
$R_2$
$R_3$
$W_a$
$W_b$
$W_c$
$W_b$'s query
$W_b$'s response is multicast to the entire tree

(c) In the *limited multicast* based scheme only $W_b$'s response is multicast to the entire tree (assuming only one allowed multicast). Therefore, $W_b$ does not know about the existence of $W_a$ and $W_c$.

**Figure 4:** ***Response Mechanisms***

its performance were acceptable, not require router support and not depend on multicast routing protocol characteristics. This solution also differs qualitatively from the previous solutions.

Consider the simple scenario where two session watchers share a common ancestor (Figure 2(a)). Suppose that one session watcher, say $W_a$ mtraces to the root, and multicasts the response. The other, $W_b$, terminates its mtrace at the common ancestor between $W_a$ and $W_b$. $W_b$ does *not* multicast its mtrace response; thus, $W_a$ is unaware of $W_b$'s existence (Figure 4(c).

Suppose now that a single fault occurs somewhere on the tree. As a result of this fault, $W_a$'s multicast response after the fault will be different from that before the fault. Suppose now that $W_b$ determines its new path to the source. If its path to the source is unchanged, then $W_b$ can infer that the fault must have occurred downstream of $r_{ab}$. If its path too has changed, then $W_b$ can infer that the fault must have occurred between $r_{ab}$ and the source. Finally, if $W_a$'s path to the source is unchanged, but $W_b$'s has changed, then $W_b$ infers that the fault must have occurred on some router between itself and $r_{ab}$.

There are two noteworthy features of limited-multicast based fault isolation. First, its fault isolation capability is *asymmetric.* While $W_b$ can infer the location of the fault, $W_a$ may not be able to infer anything based on received mtraces alone. The directed multicast based scheme has a similar feature. One obvious fix is to design a separate mechanism by means of which $W_b$ *informs* $W_a$ of the location of an affected fault. We do not consider the design of such a mechanism in this paper. Second, suppose that $W_d$ and $W_b$ share a common ancestor $r_{bd}$ which lies between $r_{ab}$ and $W_b$. In this case, assuming only $W_a$ multicasts its response, *both* $W_b$ and $W_d$ terminate their mtraces at $r_{ab}$. This is because neither session watcher knows the existence of the other. This scheme, even in steady state, can result in redundant monitoring of tree segments unlike the previous schemes presented above.

How does $W_a$ independently determine that it needs to multicast its response? For simplicity, we separate this question into two related questions: How *many* session watchers should multicast their response? *Which* of them should multicast their response? For now, assume a somewhat simplistic answer to the first—that some fixed fraction (say 5% or 10%) of the session watchers multicast. An approximate count of the number of session watchers may be known either from RTCP reports, or other estimation techniques [8]. We will revisit the scaling implications of this assumption in Section 3. As for the second question, the heuristic we use ensures that session watchers with higher values of current hop limit are the ones that multicast their responses.

The details of the limited multicast scheme are similar to that of subcast. We refer the reader to [23] for a description of the technique.

The limited-multicast based fault isolation technique biases those session watchers with higher hop-limits towards multicasting. In general, we would expect that the fault isolation error of this scheme is higher than that of subcast, and other schemes. Furthermore, the overhead of this scheme is also likely to be higher than that of subcast. Some interesting questions, which we explore in Section 3, include: Is there a regime in which the overhead of limited-multicasts is acceptable? How much worse is it than schemes that rely upon router assist?

Isolating faults in the limited-multicast approach is similar to that in the subcast approach (Section 2.1). In this case as well, $W_a$ can determine, at any instant, the session watchers $\mathcal{W}_a$ who collectively monitor segments of $W_a$'s path to the source. Armed with this information, the techniques that $W_a$ uses to isolate lossy links, or routing changes and link failures are as described before. The only difference, in this case, is that $W_a$ determines $\mathcal{W}_a$ from the multicast responses it has received.

## 2.5 Discussion

We now turn to some practical considerations and limitations of our proposed schemes.

In many of our schemes, the load on a router is proportional to the fan out of the multicast tree at that router. Thus, if there are $k$ tree links incident on a router, $k-2$ mtrace requests will terminate at that router in the subcast-based scheme. The processing overhead imposed by these messages is a function of $T$. We expect that, in practice, $T$ will be on the order of tens of seconds. As such, we do not expect that processing these multicast traceroute messages will add significantly to router overhead.

Our approach to fault isolation in multicast distribution trees assumes a relatively simple network primitive—the multicast traceroute. Unfortunately, *mtrace* may be administratively disabled in some parts of the network. In these cases, the affected receivers will clearly lack fault isolation capability. Other receivers will be able to isolate faults albeit with possibly reduced fidelity. Alternate techniques such as MINC (Section 4) might also be applicable in such cases.

All the results we show in later sections are for a single-fault model. Note that when multiple simultaneous faults occur, the faulty router identified using the single-fault model can be incorrect. Also, when multiple simultaneous faults are assumed the fault isolation error increases. We refer the reader to [23] for the fault isolation algorithm for the single and multiple fault models. Evaluation of fault isolation error in the multiple-fault scenario is the subject of future work.

The Internet multicast routing infrastructure continues to evolve. Proposed protocols, like BGMP [14] and Express [11], may be deployed as solutions for inter-domain multicast routing. We believe that these developments will not invalidate the work presented in this paper. The multicast traceroute has continued to evolve well with the addition of new kinds of routing protocols, and we see no reason why it will not do so in the future.

Finally, note that our mechanism for inferring the location of a route change can only pinpoint an on-tree router responsible for the change. In fact, the route change may have originated at a distant router; our techniques cannot infer this. However, knowing the on-tree router, an administrator can use SNMP-based techniques to unravel the cause of the fault.

## 3. EVALUATION OF FAULT ISOLATION APPROACHES

In this section, we evaluate the performance of the fault isolation schemes described in Section 2. First, we analytically examine the performance of fault isolation for *$n$-ary* trees—complete trees with a fanout of $n$ at each internal node. From this analysis, we gain some insight into the scaling behavior of various approaches, as well as their behavior under packet loss. We then examine the performance of these fault isolation schemes for irregular trees, just to verify whether the qualitative behavior revealed by analysis holds over a wider range of trees.

### 3.1 Analytic Evaluation

In an earlier section (Section 2.1), we introduced metrics that determine the performance of our fault isolation schemes. These metrics include maximum and average overhead, and the expected error. In this section, we obtain expressions for these metrics given complete $n$-ary trees with depth $d$ and session watchers located at the leaves of the tree. We study the variation of these metrics as a function of the number of session watchers, in order to get a sense of the scaling properties of the various fault isolation schemes.

To more clearly distinguish the impact of tree characteristics and the impact of loss, we derive two sets of expressions for overhead. The first set considers an idealized steady-state, non-lossy situation. In this situation, the performance of the approaches is a function of the mechanisms they use, and the tree characteristics. The resulting expressions impose a lower bound on the overhead due to these approaches. The second set considers the impact of mtrace request or response loss on overhead. As described before, message loss can trigger some session watchers to redundantly monitor links, increasing overhead. We do not consider the impact of loss on fault isolation error; while packet losses could render some links briefly unmonitored, the effect of such transients is harder to quantify.

Because the scoped multicast scheme is less amenable to exact analytic evaluation, we only consider the other three schemes in this section. Intuitively, we expect the average overhead of scoping to be no more than twice that of subcast, for regular $n$-ary trees. We discuss the performance of scoped multicast using simulation in Section 3.2. In what follows, we briefly outline the steps in deriving analytic expressions for subcast, directed multicast and limited multicasts. Then we discuss the results revealed by our analyses.

*Methodology and Assumptions*

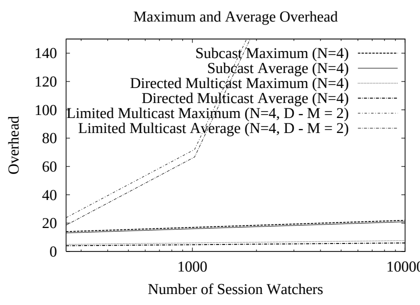
In the subcast based approach, within a window of time equal to the probe interval, we would see exactly one mtrace request per link in the steady state. The number of responses on a link is equal to the number of subcasts down each node on the link's path to the source on the multicast tree. The number of subcasts at each node on the $n$-ary tree is $n-1$. From these observations, we can compute the maximum and average overhead for the subcast approach on an $n$-ary tree (Figure 5). Notice that both of these terms are $O(nd)$. The maximum overhead occurs on the link closest the leaves. Finally, the fault isolation error is zero for subcast the subcast approach for a regular $n$-ary tree.

The overhead computation for directed multicast is very similar to that for subcast. The maximum number of response messages on any link is $d$ for the regular $n$-ary tree. More generally, the overhead for directed subcast is $\frac{1}{n}$ that of subcast (Figure 5). Intuitively, this is easy to see: at each internal node in the tree, directed subcast results in 1 message compared to $n$ in the subcast scheme. Computing the extra overhead in the presence of losses is similar to that for subcast.

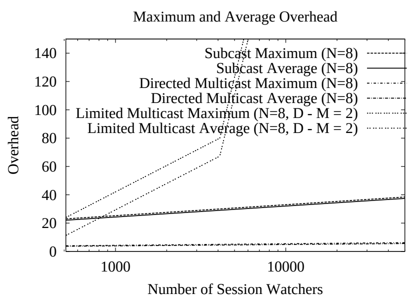
Deriving the fault isolation error is slightly non-trivial. Our computation of expected error is based on the following observation. For directed multicast, the fault isolation error at any router at height $h$ is $h$ if the router is not the leftmost child of its parent, else it equals that of its parent. This follows from our assumption that the leftmost session watcher in any subtree is the one that mtraces beyond the root of the subtree. The result is the involved expression for expected

| *Protocol* | *Overhead* | | *Fault Isolation Error* |
|---|---|---|---|
| | Maximum | Average | |
| Subcast | $(n-1)d+2$ | $2+\frac{dn^{d+1}-(d+1)n^{d}+1}{n^{d}-1}$ | 0 |
| Directed Multicast | $d+1$ | $d(\frac{n^{d+1}-n^{d}}{n^{d+1}-n})$ | $(\frac{n-1}{n^{d+1}-1})(d(d+1)+(d+1+\frac{1}{n-1})(\frac{n^{d+1}-1}{n-1}-d-1)-\frac{dn^{d+2}-(d+1)n^{d+1}+d}{(n-1)^2})-1$ |
| Limited Multicasts | $n^m+2n^{d-m+1}$ | $(\frac{n-1}{n^{d+1}-n})(\frac{n^{m+1}-n}{n-1}+n^m(d-m)+n^m(\frac{n^{d+1}-n}{n-1})+\frac{2n^m}{n-1}[(d-m)n^{d-m+1}-(d-m+1)n^{d-m}+1])$ | $(\frac{n-1}{n^{d+1}-1})(\frac{n^m-1}{n-1}+(d-m+1)^2n^m+\frac{(d-m)n^{d+2}-(d-m+1)n^{d+1}+n}{(n-1)^2}-\frac{(d-m)(d-m+1)n^m}{2})-1$ |

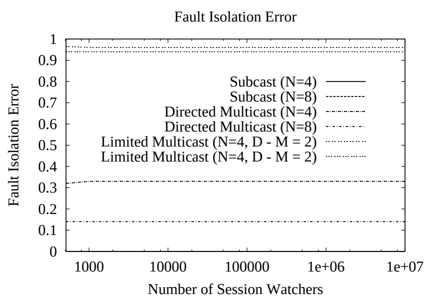
**Figure 5: Analysis of Fault Isolation Error and Overhead**



(a) Overhead, $n = 4$



(b) Overhead, $n = 8$



(c) Fault Isolation Error

**Figure 6: *Maximum/Average Overhead and Fault Isolation Error***

error shown in Figure 5.

To study the overhead of limited-multicast based approach, we fix the number of allowed multicasts at $n^m$, for some integer $m$. Then, all mtrace requests that go beyond a distance of $d-m$ are multicast. The total overhead is determined by the multicast queries, multicast responses, the queries whose responses are unicast, and responses to these queries. There is exactly one multicast query on all links upto a depth $m$. Beyond $m$, there are only $n^m$ queries of length $d-m$. Figure 5 describes the maximum and average overhead of the multicast-based scheme. The maximum overhead occurs on the link at a depth $m$ and is $n^m + 2n^{d-m+1}$. The average overhead on any link on the multicast tree is dominated by $n^m$, the number of queries whose responses are multicast.

Fault isolation error for all faults at nodes at a height less than $m$ is zero. For all other nodes the fault isolation error ranges from 0 to $d-m-1$. The expected fault isolation error is described in Figure 5.

### *Results*

To illustrate the results of our analysis, we plot the expressions in Figure 5as function of the number of session watchers in the $n$-ary tree, for different values of $n$. This methodology helps us understand the scaling behavior of the various approaches, and helps us distinguish the impact of tree characteristics and that of loss.

Figures 6(a) and 6(b) plot the overhead as a function of the number of session watchers. The overhead of both subcast and directed multicast scales logarithmically with the number of session watchers. This is clearly desirable scaling behavior. Furthermore, the maximum and average overheads for these schemes are very nearly indistinguishable. The maximum and average overhead for limited-multicast *increases linearly* with the number of session watchers—we keep $d-m$ fixed, which fixes the fraction of session watchers that are allowed to multicast their responses (for $n = 4$ this corresponds to about 6% of the session watchers multicasting, for $n = 8$ it corresponds to about 1.5%). This is definitely undesirable scaling behavior. However, an interesting observation is that for the larger fanout, there is a regime (for group sizes less than about 700) where the average overhead for limited-multicast *is less than that of subcast.* This suggests that subcast is more sensitive to fanout, a subject we explore in Section 3.2.

Figure 6(c) plots the expected fault isolation error as a function of the number of session watchers. The error for subcast is zero. A surprising result is that, for $n$-ary trees, the expected error of directed multicast is asymptotically constant. Furthermore, the value of this constant appears to decrease with increasing $n$. Furthermore, the average error for the limited-multicast scheme is independent of the number of session watchers, and is less than 1. There is a tradeoff between the number of allowed multicasts and the average error. By keeping $d - m$ constant, as we have done, we have kept the fraction of allowable multicasts constant, keeping the fault isolation error constant but linearly increasing overhead.

## 3.2 Impact of Tree Characteristics on Performance

In the previous section we obtained analytical expressions for overhead and fault isolation error for regular trees of various sizes. In such trees, all non-leaf nodes have the same fanout, and all leaves are at the same distance from the source. Our analysis in Section 3.1 showed that at least the subcast scheme is somewhat sensitive to fanout. Furthermore, the limited-multicast scheme relies on path length to determine which session watcher multicasts. For these reasons, we now explore the performance of these schemes across a variety of *irregular trees*. The goal of this evaluation is to understand whether, for any of the schemes, the performance changes qualitatively when going from regular to irregular trees. This evaluation also helps us examine the performance of scoped multicast based fault isolation (Section 2.3).

As before, to isolate the impact of tree characteristics on performance, we first study these schemes in an idealized, non-lossy setting. Studying the impact of loss on irregular trees would require packet-level simulations. Because these simulations do not allow us to scale beyond a few hundred session watchers, we conducted only a limited evaluation of the impact of loss. This is discussed in [23].

To study the impact of irregular fanout and non-uniform path lengths between session watchers and the source, we use the random tree generator described in [21]. The input to this generator consists of the number of leaves, the total number of nodes, and the maximum fanout of nodes in the tree. This generator produces randomly generated trees whose internal nodes have a fanout between one and the maximum fanout. Although this generator has three input parameters, it has two degrees of freedom: the fanout and the number of leaves or session watchers.

In this section, we separately study the impact of fanout and the impact of scale on our fault isolation schemes. For each dimension that we explore, we study the overhead and fault isolation error of the various schemes. Since we consider an idealized, non-lossy situation, we do not need packet level simulation to compute these metrics. Instead, for a given point in our design space (*i.e.,* given a tree), there exists a simple algorithm to compute the overhead and error for each scheme. In the graphs shown below, for each point in the space, we average the performance measures over ten randomly generated trees.

### *Scaling Behavior*

Figure 7 plots the performance of our fault isolation schemes as a function of the number of session watchers, for two different maximum fanouts.

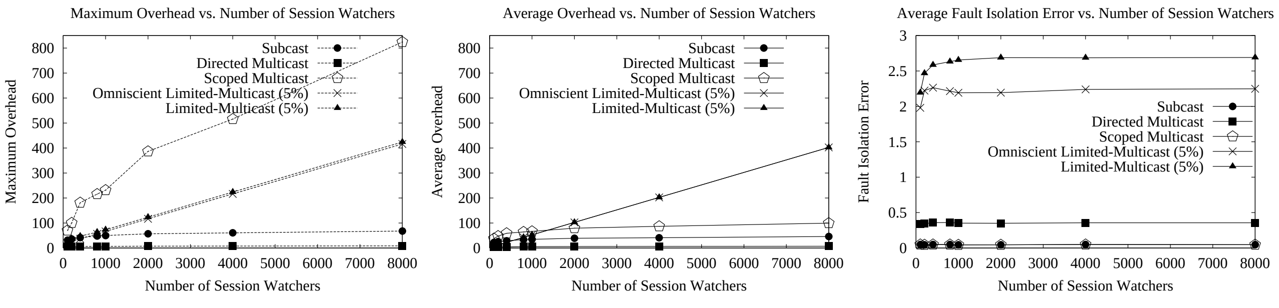
The subcast scheme exhibits slowly increasing overhead as a function of the number of session watchers. So does directed multicast. For both schemes the maximum overhead is distinguishable from the average overhead, unlike Figure 3.1. Thus, the irregularity of the tree fanout affects the distribution of link overheads, but does not qualitatively change the scaling behavior of the two approaches. As fanout increases, the disparity between subcast and directed multicast increases. This is because, for a given number of session watchers, at higher fanout the height of the tree is smaller. Directed multicast overhead is proportional to the height, hence the overhead of directed multicast is actually lower at the higher fanout. Increasing the fanout adversely impacts subcast, as we discussed in Section 3.1.

The fault isolation error of router assist schemes is also quite low, and is independent of the number of session watchers. With low maximum fanout, there are more likely to be tree segments without branches. For this reason, the expected fault isolation error for subcast is non-zero. However, with a higher maximum fanout, the expected error for subcast is nearly zero. By contrast, directed multicast has non-zero error regardless of fanout. Its expected error decreases with increasing maximum fanout. We also observed these results in our analysis (Section 3.1).

The scoped-multicast scheme behaves differently with respect to overhead than the other schemes. The average overhead of scoped multicast exhibits identical behavior to that of the subcast scheme, logarithmic scaling. This confirms our intuition that, for regular trees, the average overhead of scoped-multicast is no more than twice that of subcast. For irregular trees, however, its average overhead is more than a factor of two higher than subcast. The ratio approaches two at higher fanout. However, the maximum overhead of scoped-multicast matches that of limited-multicast at low fanout, and is higher at higher fanout. Unlike subcast, scoped-multicast responses are not limited to the subtree at whose root a session watcher's mtrace terminates. Rather, these responses can traverse links in other parts of the tree, explaining the high maximum overhead. It is unclear whether the maximum overhead for scoped multicast increases linearly or has sub-linear growth. We intend to resolve this by conducting larger scale simulations. Finally, the fault isolation error of scoped-multicast is identical to that of subcast.

The overhead of limited-multicast also increases linearly with increasing number of session watchers. This is not surprising, since the overhead is dominated by the number of allowed multicasts, which is kept at a fixed fraction of the number of session watchers. However, the maximum overhead of the limited-multicast scheme deviates more from the average than predicted by our analysis. As in our analysis, too, there is a small regime where limited-multicast outperforms subcast.

Figure 7 attempts to calibrate the performance of our limited-multicast heuristic (based on length of the current hop limit). To do this, we compute overhead and error for an *omniscient limited-multicast* scheme. In this scheme, the session watchers that are allowed to multicast their responses are chosen such that redundant monitoring of links in the tree is minimized. The omniscient scheme represents the least overhead achievable by a limited-multicast scheme. It does not attempt, however, minimize fault isolation error. As



(a) Maximum Overhead (b) Average Overhead (c) Fault Isolation Error

**Figure 7: Performance as a function of the number of session watchers, maximum fanout = 12**

the figures show, the average overhead of limited-multicast is almost identical to that of omniscient limited-multicast. Their maximum overheads differ for low degree of sharing, but for higher fanouts, even this distinction disappears.

The expected error for limited-multicast is significantly worse than either of the schemes that employ router assist. The disparity is greater than that predicted by our analysis. Interestingly, however, the expected error is independent of the number of session watchers. Also note that omniscient limited-multicast has comparable fault isolation error; at higher fanout, it has higher expected error.

To understand the practical import of our results, consider that in Figure 7, with 8000 receivers, the average overhead of limited multicast is about 400 messages. If we assume that the probe interval is 10 seconds, and the average mtrace message size is about 200, this corresponds to an overhead of 64 kbps. By contrast, the overhead for scoped-multicast is 16 kbps, subcast is 8 kbps and that for directed multicast is 1.2 kbps.

In experiments discussed in more detail in [23], the behavior of increased fanout for most schemes is as one might expect. For example, subcast overhead increases linearly with fanout, but its fault isolation error drops significantly.

### 3.3 Summary of Performance Evaluation

In summary, then, our evaluation of irregular trees largely confirms our analysis and provides some insight into the performance of scoped-multicast based fault isolation. While in some cases, the distribution of link overhead appears to be different, the qualitative behavior of the various schemes is not altered. Our three primary findings follow. First, router assist allows very desirable scaling behavior for fault isolation. Both subcast and directed multicast scale logarithmically as a function of the number of session watchers. Directed multicast is a better choice of router assist for fault isolation; its overhead is lower than that of subcast, its expected error is not significantly greater, it is relatively insensitive to fanout, and is the most loss tolerant of our schemes. Second, our carefully designed, deployable alternative, limited-multicast, works with acceptable overhead for small groups of fewer than a few hundred participants. In this regime, it sometimes has less overhead than subcast, for certain kinds of trees. Even there, however, its fault isolation error is high. This means that limited-multicast cannot be used to isolate routing changes. It can, however, be used to locate lossy links [3]. Finally, as a deployable solution—modulo the routing protocols that break TTL-scoping semantics—scoped-multicast performs well on average. The one caveat to this, however, is the relatively high maximum overhead induced by scoped-multicast.

This paper has almost exclusively concentrated on the impact of scale. However, our techniques all share several mechanisms which can impact the *latency* of fault isolation. For example, session watchers initially "hunt" in order to find their turnaround router. Furthermore, all schemes have mechanisms that eventually react to receiver leaves, response losses and session watcher failures. These mechanisms determine how long routers in the tree go "unmonitored" and impact the temporal fidelity of fault isolation. Evaluating this is the subject of future work.

## 4. RELATED WORK

Little prior work has been done in automated multicast tree fault isolation. Currently, when receivers in a multicast session observe problems, system administrators manually send several ad-hoc mtrace requests between arbitrary receivers and sources to detect faults. The *mhealth* tool [18] effectively conducts mtraces from every receiver to the source, and does provide similar functionality to our algorithms, albeit in a centralized, non-scalable fashion.

Of most relevance to our work on fault isolation is MINC. In MINC, bottleneck links and the multicast logical tree topology are inferred by correlating packet losses observed at receivers. Such inference is either based on rigorous statistical techniques [2] or on heuristics [21]. By correlating loss *patterns* at each receiver, both approaches can determine which receiver (or sets of receivers) share a common ancestor, thus determining the *logical* tree topology. The same technique can be used to determine the location of the bottleneck link in this logical topology. Unlike our approach, these approaches do not rely on any diagnostic functionality (such as mtrace). However, in theory, MINC can be used

[3]Recall that fault isolation error is only relevant to locating failed routers or the origin of a routing change.

in conjunction with our approach to improve fault isolation accuracy or to "cover" regions without mtrace support.

Multicast traceroute has been used in protocols for reliable multicast to determine tree topologies [15, 16]. In these schemes, receivers use multicast traceroutes to discovers the nearest peer on the other side of a bottleneck link.

Of relevance to our work are techniques for proactive fault isolation. One such approach [12] *learns* normal behavior of the network by developing estimates for router state, then detects and flags deviations from these estimates. Such a system can detect abnormal behavior before a fault actually occurs. Similar learning techniques can be employed in our algorithms as well.

Some of our techniques for robust coordination between receivers draw inspiration from earlier work on techniques for multicast application design [19]. Specifically, our periodic probing combined with subcast or multicast responses is an instance of the *announce-listen* technique for robustness. This also enables *shared learning* of the location of faults.

## 5. CONCLUSIONS

In this paper, we explored the design space of fault-isolation schemes for large single-source multicast trees, given a receiver-driven path probing primitive. We explored router-assist for selective forwarding of probe responses, and considered carefully designed deployable alternatives. Our technique for fault isolation uses path probe history, and shared learning of probe results to scale well and infer the location of route changes, router failure, or lossy links.

We find that, while one deployable alternative exhibits some attractive scaling behavior, schemes that employ router assist outperform deployable alternatives. Our work represents an important contribution to the debate over selective forwarding strategies for multicast. So far, the need for such strategies had just been considered for reliable multicast. We show that selective forwarding can qualitatively change scaling behavior of fault isolation.

### *Acknowledgements*

The authors would like to thank Vern Paxson and Mark Handley for comments on earlier versions of this work. Comments from the reviewers and from our shepherd Ellen Zegura greatly helped improve the quality of the presentation.

## 6. REFERENCES

[1] Cabletron Systems. Spectrum. http://www.cabletron.com/spectrum/.

[2] R. Caceres, N. Duffield, J. Horowitz, D. Towsley, and Tian Bu. Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation. In *Proceedings of the IEEE Infocom*, New York, NY, March 1999.

[3] Cisco Systems. Netflow. http://www.cisco.com/warp/public/732/netflow/index.html.

[4] A. Costello and S. McCanne. Search Party: Using Randomcast for Reliable Multicast with Local Recovery. In *Proceedings of the IEEE Infocom*, New York, NY, March 1999.

[5] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE Transactions on Networking*, 4(2), April 1996.

[6] D. Estrin and D. Farinacci. Bi-directional Shared Trees in PIM-SM: Protocol Specification. Internet draft. draft-farinacci-bidir-pim-01.txt.

[7] W. Fenner and S. Casner. A Traceroute Facility for IP Multicast. Internet Draft, June 1999.

[8] T. Friedman and D. Towsley. Multicast Session Membership Size Estimation. In *Proceedings of the IEEE Infocom*, New York, NY, March 1999.

[9] M. Handley, I. Houvelas, and L. Vicisano. A New Proposal for Bi-directional PIM. Internet draft. draft-kouvelas-pim-bidir-new-00.txt.

[10] Hewlett Packard. HP Openview. http://www.hp.com/openview/index.html.

[11] H. Holbrook and D. Cheriton. IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications. In *Proceedings of ACM SIGCOMM 1999*, Boston, MA, September 1999.

[12] Cynthia S. Hood and Chuanyi Ji. Proactive Network Fault Detection. In *Proceedings of the IEEE INFOCOM*, Kobe, Japan, April 1997.

[13] IBM. Netview for AIX. http://www.networking.ibm.com/nv6/nv6prod.html.

[14] K. Kumar, P. Radoslavov, C. Alaettinoglu, D. Estrin, M. Handley, and D. Thaler. The MASC/BGMP Architecture for Wide-Area Multicast Routing. In *Proceedings of ACM SIGCOMM 1998*, Vancouver, British Columbia, September 1998.

[15] B. N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves. Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation. In *Proc. Sixth ACM International Multimedia Conference (ACM Multimedia 98)*, September 1998.

[16] D. Li and D.R. Cheriton. OTERS (On-Tree Efficient Recovery using Subcasting) A Relible Multicast Protocol. In *Proc. Sixth IEEE International Conference on Network Protocols (IEEE ICNP'98)*, October 1998.

[17] J. C. Lin and S. Paul. A Reliable Multicast Transport Protocol. In *Proceedings of the IEEE Infocom*, San Francisco, CA, March 1996.

[18] D. Makosfske and K. Almeroth. MHealth: A Real-Time Multicast Tree Visualization and Monitor ing Tool. In *Network and Operating System Support for Digital Audio and Vid eo (NOSSDAV)*, New Jersey, USA, June 1999.

[19] S. McCanne. Scalable MultiMedia Communication with Internet Multicast, Lightweight Sessions and the Mbone. Technical Report CSD-98-1002, University of California, Berkeley, 1998 1998.

[20] C. Papadopoulos, G. Parulkar, and G. Verghese. An Error Control Scheme for Large-scale Multicast Applications. In *Proceedings of the IEEE Infocom*, San Francisco, March 1998.

[21] Sylvia Ratnasamy and Steven McCanne. Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-end Measurements. In *Proceedings of the IEEE Infocom*, New York, NY, March 1999.

[22] A. Reddy, D. Estrin, and R. Govindan. Large-Scale Fault Isolation. *To appear, IEEE Journal on Selected Areas in Communications*, 2000.

[23] A. Reddy, R. Govindan, and D. Estrin. Fault Isolation in Multicast Trees. Technical Report TR-00-723, Computer Science Department, University of Southern California, Los Angeles, February 2000.

[24] SUN Microsystems. Solstice. http://www.sun.com/solstice/.