

# Large-Scale Fault Isolation

Anoop Reddy, Deborah Estrin, and Ramesh Govindan

**Abstract**—Of the many distributed applications designed for the Internet, the successful ones are those that have paid careful attention to scale and robustness. These applications share several design principles. In this paper, we illustrate the application of these principles to common network monitoring tasks. Specifically, we describe and evaluate 1) a robust distributed topology discovery mechanism and 2) a mechanism for scalable fault isolation in multicast distribution trees. Our mechanisms reveal a different design methodology for network monitoring—one that carefully trades off monitoring fidelity (where necessary) for more graceful degradation in the presence of different kinds of network dynamics.

**Index Terms**—Network fault diagnosis, network mapping, multicast.

## I. INTRODUCTION

TRADITIONAL client-server applications still dominate Internet traffic. Nowadays, however, we are witnessing the emergence of truly distributed applications and services on the Internet. Examples of such applications include Web document caching hierarchies [2], audio and video multicast transmissions, and wide-area shared workspaces [18]. Network dynamics (transient congestion, route changes) significantly affect the design and performance of these inherently multipoint applications. To these applications, network dynamics manifest themselves as *faults*—lost or delayed transmissions. An important, but unexplored, research area is the design of a large-scale infrastructure for *isolating* such faults—locating the routers responsible for these faults.

To motivate the need for large-scale fault isolation, we first consider a specific class of applications: those based on IP multicast [14]. Multicast [14] is a network layer primitive for group communication in the Internet. Today's IP multicast service is implemented by a *virtual* network, the *Mbone* overlaid on the Internet. Multicast applications have now been in use for many years [41]. For example, real-time conferencing applications transmit packetized audio and video from one or more senders to all receivers of a multicast group [40], [39], [55]. Another class of applications allows a group of conference participants shared, dynamically annotatable views of documents and text [18], [22]. The characteristics of individual *sessions* of these applications may vary widely, ranging from tens to thousands of participants.

Compared to a single session of a point-to-point application, users in a multicast session are more likely to be collectively affected by network *dynamics*. In a best effort datagram network

such as the Internet, two kinds of dynamics are visible to an application: *variation in packet delay* and *packet loss*.<sup>1</sup> An infrastructure that allows users or network administrators to *locate* the router that is responsible for a particular observed dynamic can improve the manageability of today's networks.

What are the causes of these dynamics? Increases in router load can result in increased queuing; applications might see varying packet delay [46]. Under transient congestion situations, routers may drop packets, resulting in application-perceived loss [30]. In extreme cases, underprovisioned links and routers can result in sustained congestion. Path changes between sender and receiver(s) constitute a second class of causes. There are many reasons for such path changes: router or link failure, router misconfiguration [33], routing instability caused by incorrect implementation or unexpected software interactions [38], and other kinds of routing pathologies [45]. This list of causes is intended to be illustrative, not exhaustive. In fact, enumerating all possible causes for a perceived dynamic—such as packet loss—may be difficult.

These dynamics may affect different applications in different ways. For example, the throughput achieved by long file transfers may be affected very little by occasional packet loss. At the other end of the spectrum, network dynamics can significantly impact multicast-based audio and video conferencing applications. For example, several artifacts have been observed in audio conferencing sessions: several consecutive sentences uttered by a participant are inaudible; some participants temporarily lose connectivity with the rest of the group, usually caused by a routing change; and there is asymmetric connectivity between participants, where one participant can hear another, but not vice versa.

Above, we listed several causes of network dynamics. We use the term *fault* to refer to one instance of such a cause at a single router; this router is said to represent the *location* of the fault. Thus, one example of a fault is transient congestion at a router. In this case, that router itself is said to constitute the fault's location. Another example is a route change to a particular destination. In this case, the router that originally triggered the route change is said to represent the fault's location.

Some faults may have distant causes. One example is router failure caused by routing table overflow. The origin of the overflow may have been a distant router incorrectly injecting a large number of routes into the network. By our definition, the location of the fault is the failed router, not the distant router that caused the failure. In this situation, our techniques help isolate the first-order cause of the application-perceived dynamic. We expect that the original cause (the distant router) can be isolated using other tools (e.g., postmortem analysis of

Manuscript received March 27, 1999; revised November 30, 1999. 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.

The authors are with the USC/Information Sciences Institute, Marina del Rey, CA 90292 USA (e-mail: areddy@isi.edu).

Publisher Item Identifier S 0733-8716(00)02758-X.

<sup>1</sup>Complete loss of connectivity can be described as infinite packet loss, or infinite delay. Other network dynamics, such as packet duplication and packet reordering, are also visible to applications. Investigating fault isolation mechanisms for these is the subject of future work.

the failed router, together with SNMP monitoring history of routers in the vicinity).

A single fault may affect several application sessions. Thus, transient congestion at a router may affect all application flows passing through that router. Similarly, a route change can reroute all application flows whose destinations match the route. Conversely, since the effect of a fault depends on the application, a single fault may not affect any application flows at all. Thus, a transient queue buildup at a router may not affect any application flows at all if, for example, all flows correspond to file transfer applications.

We have introduced the terms fault and fault isolation, and defined these terms by example. The precise definition of a fault depends on the class of applications for which a fault isolation mechanism is designed. Section III defines faults a bit more carefully for multicast applications.

Having defined faults, we define *fault isolation* to be the process of inferring a fault's location. The starting assumption of this paper is that such fault isolation is a desirable capability in large networks, especially for application sessions that span a wide topological region. In practical terms, a fault isolation infrastructure can help designers debug their applications, and network administrators efficiently track customer complaints about observed network pathologies.

One way to isolate faults is to probe the network after the fault has been detected. For example, a multicast "traceroute" [56] probe can elicit packet loss information on the path from a sender to a receiver. This information can locate the router(s) responsible for unusual packet loss. Such post-facto probing may be insufficient for fault isolation; for example, the multicast traceroute cannot deduce information about the origin of a route change since it is equally affected by the change.

To overcome this limitation, a fault isolation system may collect information from one or more nodes in the network, then use this *history* of collected information, together with reactive probing, to isolate faults. This paper focuses on the design of such a fault isolation infrastructure in large networks.

Commercial network monitoring systems fit the description of the previous paragraph. For example, many existing commercial network monitoring products [25], [23], [53], [4] centrally collect information from standard (SNMP [42] MIB's) or proprietary (NetFlow [9]) interfaces. They then use this information to detect router failures, congested links, and other anomalous situations.

We believe that such systems may not be able to isolate faults in, for example, an audio conferencing application. First, monitoring routers may not be sufficient for correlating an application-perceived behavior (e.g., packet delay variation) with router activity (e.g., route change, or queue buildup). For example, given that a route change has occurred at a router, it may be impossible to infer the affected application flows. This is because routers do not, for scaling reasons,<sup>2</sup> maintain per-flow state for *every* flow transiting the router. Furthermore, network dynamics affect applications in different ways. Thus, depending upon the application flows transiting a router, some

network dynamics at a router may not affect applications at all. Second, systems that collect data in a central node cannot obviously scale to the size of the Internet. Such systems may be sufficient for a mid-sized ISP with a few hundred routers; they are clearly insufficient for the larger Internet with several hundreds of thousands of routers [21].

Even if we could scale to the size of the Internet by distributing the SNMP data collection—and we do not mean to imply that this is the right design for fault isolation—the design of such a distributed monitoring system is nontrivial. Why? An implicit assumption we have made in the preceding sections is that the software system used to locate the source of network dynamics is *in-band*, i.e., it uses the self-same network to collect information. From this assumption, it follows that the distributed monitoring system *itself* is subject to the same network dynamics. In some cases, however, especially in the current instantiation of IP multicast (the Mbone), the multicast and unicast infrastructures may not be congruent. In these situations, multicast monitoring information may make use of "out-of-band" information provided by unicast. Our mechanisms do not leverage this, because it is unclear whether this is an artifact of today's deployment methodology, or a longer term fact.

In this paper, we do not presume to design a large-scale fault isolation infrastructure—this is, in many ways, the Holy Grail of all of network management research. However, we have already pointed out two key problems in the design of such an infrastructure: that it must *scale* to the size of the Internet, and that it must be "robust" to network dynamics. Experience with the design of Internet protocols and distributed multicast-based applications has revealed a few related techniques [41] for robustness and scalability. We describe these techniques below.

1) *Announce-Listen*: In this *robustness* technique, every participant periodically announces a summary of its state in the distributed computation [8]. This announcement is multicast to all other participants; receivers update their state according to these summaries. There is usually no other explicit handshake between participants. This technique helps deal with group dynamics as well as network dynamics.

2) *Empirical Adaptation*: This *robustness* technique requires applications to adapt their behavior to the perceived state of the network. A good example is TCP's send-window adaptation to network congestion [30] based on lost packets. A similar example may be found in the design of a protocol called RLM for the transmission of layered video [40].

3) *Shared Learning*: See [40]. Related to both the above techniques, this *scaling* technique allows participants of a distributed computation to learn either about the state of the network, or about some other aspect of the distributed application, from other participants. A form of shared learning may be found in some reliable multicast protocols (e.g., SRM [18]).

In this paper, we demonstrate the application of these principles to two network monitoring tasks:

- 1) the robust distributed collection of network maps; and
- 2) scalable monitoring of multicast distribution trees in order to isolate certain types of faults.

The following sections describe these two applications in greater detail.

<sup>2</sup>Routers may maintain *some* per-flow state, e.g., space for policing misbehaved flows [17].

## II. ROBUST DISTRIBUTED MAPPING OF LARGE NETWORKS

A network map is an important component of any network fault isolation system. By a network map, we mean a graph whose nodes represent routers and whose links represent router adjacencies. Many network monitoring systems provided *annotated* views of network maps. In these views, individual routers may be annotated with aggregate packet loss statistics, or outage histories, average delays, and so on.

Network maps cannot directly be used to isolate application-perceived faults—since routers maintain aggregate state, knowing that a fault has occurred at a router does not enable us to infer which applications might be affected by that fault. Nevertheless, router-level maps play an important role in fault isolation. They provide the user or network administrator with a topological *context* for the fault. By looking at a map, we may be able to determine, for example, that a fault is located at an egress router out of a cloud. Such information may make it easier to understand why a fault has occurred (in this case, for example, because the egress router is a potential bottleneck).

### A. A Distributed Mapping Scheme

Some network monitoring systems [13] collect network maps using SNMP. We use the term **surveyor**<sup>3</sup> to denote any software entity that collects network maps. A surveyor could use the OSPF MIB's [15] neighbor table, or the RIP MIB [19] peer table for discovering routers. Approaches such as those that use a single surveyor do not scale beyond a few hundred routers. Large networks exhibit significant dynamics such as network partitions or host failures. Thus, network maps collected only once and available from a single surveyor can be rendered incorrect or unavailable during network dynamics.

We now describe how these drawbacks may be overcome using a distributed mapping scheme. In our scheme, one or more surveyors are deployed within the network. Each surveyor uses a simple recursive neighbor discovery algorithm to discover the portion of the network in its immediate vicinity. Surveyors then *coordinate* with each other to ensure that they do not redundantly map sections of the network.

Before we describe how our distributed mapping scheme works, we introduce some terminology and notation. Each surveyor assigns to every router in the network a *distance*. One metric, but by no means the only one, for distance is the number of hops in the shortest path from the surveyor to that router. A surveyor's *range* is the set of routers closer to it than to any other surveyor. We denote surveyors by integer scripts associated with  $S$ . We say that a router *belongs* to a surveyor  $S_i$  if it is contained in  $S_i$ 's range.

The goal of our distributed mapping scheme is to ensure that *each surveyor maps that portion of the network that corresponds to its range*. At any instant during the execution of our distributed mapping scheme, the map of the network is distributed between different surveyors. Note that the definition of a surveyor's range is *not* static. For example, when one surveyor fails, the ranges of neighboring surveyors might increase. Similarly, when the network partitions, a router that previously belonged to one surveyor might now belong to an entirely different surveyor.

Clearly, in order to compute its range, a surveyor must learn about the routers discovered by other surveyors. We now outline the *coordination protocol* that enables robust, distributed, range determination. For this protocol, surveyors use a single well-known multicast group. At any point in its lifetime,  $S_i$  has discovered a certain section of the network topology within its vicinity. The routers in this section constitute the surveyor's *computed range*. We denote  $S_i$ 's computed range by  $\mathcal{R}_i$ . When surveyor  $S_i$  starts up, it initializes  $\mathcal{R}_i$  to contain only one element—itsself. The surveyors protocol actions are outlined below.

- 1) With a frequency that is a function of the size of its computed range,  $S_i$  *expands* its current computed range by a small increment. To do this, it probes routers on its range boundary to detect neighbors that are not in  $\mathcal{R}_i$ .  $S_i$  incorporates these newly discovered routers into  $\mathcal{R}_i$ .
- 2)  $S_i$  periodically *multicasts* its computed range on the coordination channel. Along with its range, the surveyor also advertises its distance to routers in its range.
- 3) When it hears a range  $\mathcal{R}_j$  advertised that intersects with its own,  $S_i$  excludes some of the common routers from its own range. The extent of this *backoff* is a function of  $S_i$ 's and  $S_j$ 's distances to routers in the overlapped region.  $S_i$  then advertises its adjusted range on the coordination channel.

In item 1 above,  $S_i$  multicasts its computed range periodically. Thus, every other surveyor  $S_j$  learns the discovered neighborhood of this surveyor. Using this information in item 3,  $S_j$  can independently determine whether a router within its own computed range (if any) does not belong to its range:  $S_i$ 's distance to such a router would be less than  $S_j$ 's own distance to that router. After some number of iterations, every  $S_i$ 's computed range converges to its range. This sequence of actions is an application of the *announce–listen* technique.

However, the distributed computation itself does not converge. In item 1 above, each surveyor periodically expands its computed range. This is an example of the *empirical adaptation* technique. By *continually* attempting to expand its computed range, each surveyor automatically adapts to the failure of other surveyors, the partitioning of the network, etc. Illustrations of how the protocol adapts to various network dynamics can be found in [49].

We now summarize several important properties of our distributed mapping scheme:

- In item 1 above, each surveyor expands with a frequency that depends on the size of its computed range. In this way, surveyors with larger ranges expand more slowly. This makes intuitive sense—the load on a surveyor is a function of the number of nodes in its range.
- Our distributed mapping scheme is highly robust to surveyor failures and network partitions. In fact, as long as one surveyor is active, a network map will eventually be computed, albeit more slowly. If the coordination channel is noisy, there may be some overlap between the computed ranges of surveyors.
- Our scheme does not require coordinated surveyor placement. Surveyors, regardless of their placement, eventually rendezvous with each other on the coordination channel.

<sup>3</sup>Not to be confused with the Surveyor project [32].

However, nonoptimal surveyor placements *can* result in the unequal surveyor computed ranges, and longer convergence after surveyor failures.

- The protocol has a single *mode* of operation. Network dynamics are not treated as a special phase of protocol operation. This feature simplifies the operation of the protocol. Furthermore, network dynamics such as route changes themselves do not require recomputation of the map, since a router adjacency may continue to exist despite a route change.

### B. Distributed Mapping Design Issues

Recall that both *announce-listen* and *empirical adaptation* are robustness techniques. What techniques might we use to scale the protocol to larger networks? Some degree of scaling in map collection is achieved by distributing the collection of maps among many surveyors. However, our protocol still requires each surveyor to periodically multicast its entire range (part 2 of the protocol description). If this periodicity is fixed, larger networks incur significant overhead. Instead, our protocol *adapts* the periodicity of surveyor multicasts to the size of the network. This represents a tradeoff between protocol overhead and convergence after failure. Two additional scaling techniques are possible. First, each surveyor multicasts only the *boundary* of its current computed range. This can reduce the overhead of range advertisements, but can impact the stability of the protocol. Second, each surveyor need only advertise its range to neighbors. This can be accomplished by scope limiting the advertisements to reach only the neighbors. We are currently working on these approaches.

Our distributed mapping scheme can form the basis of a large-scale monitoring system. Each surveyor can continuously monitor SNMP interfaces of routers within its computed range. With this approach, each node is monitored by its nearest surveyor. The design of protocols that access monitoring information [47] from surveyors is beyond the scope of this paper. Clearly, this approach may not be applicable to heterogeneous networks like the Internet, where a surveyor within one administrative domain might not be able to access SNMP MIB's at routers in other domains. However, we believe that our approach can be extended to heterogeneous networks by allowing federations of surveyors in one administrative domain to exchange *filtered* information about each others' networks.

In our description so far, we have assumed that surveyors can access SNMP interfaces in order to discover router adjacencies. SNMP is not the only technique available for determining router adjacencies. On the Mbone, for instance, the DVMRP\_ASK\_NEIGHBORS IGMP [57] message might be used for this.

### C. Performance Evaluation and Implementation

We have simulated our distributed mapping scheme in *ns* [1]. Fig. 1 describes the behavior of our algorithm on a randomly generated 300-node transit-stub topology [58]. On this topology, we randomly placed  $N = 4, 6, 8, 10$  surveyors. For each surveyor placement, we generated random surveyor failures with the time between one surveyor failure and the next chosen uniformly from  $[0 \dots T]$ , for  $T$  varying from 100 to

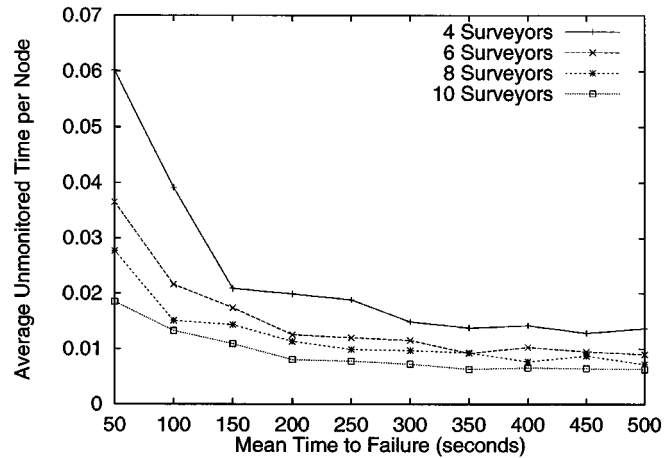


Fig. 1. Surveyor adaptation in a failing network. The graph plots the fraction of simulation time a node was unmonitored versus the mean time to failure in a random 300 node transit stub topology.

900 in steps of 100 units. The duration of each simulation was 1500 units. The x axis on Fig. 1 plots the average interfailure time ( $T/2$ ). The y axis shows the fraction of simulation time during which a node did not belong to any surveyor's range (i.e., the node was "unmonitored"), averaged over all nodes. As expected, even with only four surveyors and a surveyor failing every 50 s, a node was unmonitored only 6% on average. Furthermore, as the number of surveyors increased, or as the average interval between two surveyor failures increased, this fraction reduced dramatically to less than 2%. Clearly, more extensive simulations are necessary to understand the impact of surveyor placements, the impact of packet losses, etc. At the very least, however, these simulations validate our basic design and illustrate some features of the algorithm.

We have currently implemented this distributed mapping scheme for the Mbone. As described above, surveyors use the DVMRP\_ASK\_NEIGHBORS to determine router adjacencies. They then multicast their computed ranges on well-known multicast groups. Our implementation has revealed several interesting Mbone artifacts: routers which are revealed in DVMRP\_ASK\_NEIGHBORS messages but do not themselves respond with such messages (presumably because there are no routes to these messages), vendor specific anomalous responses to these messages, etc. We are currently augmenting our implementation with techniques that infer router adjacencies from *mtrace* responses.

### III. SCALABLE FAULT ISOLATION FOR MULTICAST SESSIONS

This section considers the problem of fault isolation in a multicast distribution tree. We first describe the problem at hand, and consider alternative approaches. We then describe a novel receiver-driven technique for scalably monitoring a multicast distribution tree. This technique exhibits some interesting properties that enable rapid fault isolation.

#### A. Fault Isolation for Multicast Trees

In Section I we defined a network *fault* to mean a perceived increase in the delay of packet delivery, or the loss of a packet. Usual causes for faults include router failure, route changes, or

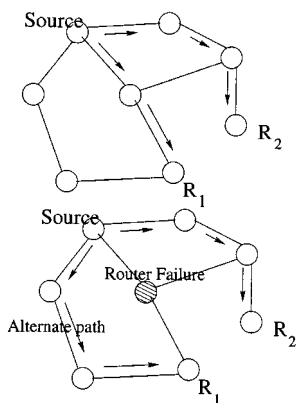


Fig. 2. An example multicast session. The effect of a router failure that results in a path change is illustrated here.

transient congestion. In this section, we consider the problem of fault isolation as applied to multicast distribution trees. Specifically, we ask the question: Given that one or more receivers on a multicast distribution tree perceive a fault, what mechanisms might we use to rapidly deduce the location of that fault?

For concreteness, consider the following scenario (Fig. 2). An audio or a video lecture is in progress, and receivers join and leave this lecture session, perhaps during the lecture. As we have described in Section I, routers inside the network construct a distribution tree from the source of the lecture to all the receivers. The tree itself is dynamic: it changes when routes change, when routers fail, or when receivers join or leave. The number of receivers is potentially large; it is not inconceivable that a significant fraction of network users might concurrently subscribe to the lecture.

Consider now the impact of a failed router. This router may result in several observable artifacts at the affected receivers. First, it may cause a burst of observable losses until the network repairs the path between the source and the affected receivers [Fig. 2(a)]. Second, packets may incur greater delay along the new path than along the old—this may result in a one-time glitch in the audio or video, especially if the delay disparity is large. Finally, if the new path traverses a congested link, receivers may observe sustained degradation in audio or video quality. Similar artifacts may be observed when congestion occurs at a router; these are particularly acute in today’s audio or video applications that do not adapt their transmission rates in response to congestion [50].

This scenario and these artifacts motivate the need for a collection of mechanisms that help users or network administrators deduce the location of an application-perceived fault. Such a capability is essential particularly when—as in the case of our multicast lecture example—a single fault can affect a large number of receivers. An explicit nongal is the ability to unravel the set of circumstances that caused a fault. For example, we are not interested in identifying the exact flows that have contributed to observed transient congestion at a router. We assume that once a fault has been located, a network administrator can use other methods (SNMP, system logs, etc.) for this purpose.

Before we discuss how to design fault isolation mechanisms, we *qualitatively* describe some requirements of these mechanisms.

- The mechanisms must *scale* to a very large number of receivers. In future networks, sessions with millions of receivers are quite likely. Today, the receivership of popular televised events easily reaches that number. It follows that the mechanisms must be designed for widely distributed receivership.
- The mechanisms must be *robust* to receiver joins and leaves. In particular, the joining or leaving of a receiver must not render the fault isolation capability inoperable at other receivers. Furthermore, when a network partition results in some receivers being unreachable from the source, the remaining receivers must still be able to isolate faults.
- It is acceptable for the fault isolation mechanisms to localize a fault to within some topological neighborhood of the actual fault location. We believe that a fault isolation mechanism cannot simultaneously achieve perfect *spatial fidelity*, scale, and robustness (Section V).
- It is acceptable for the mechanisms to altogether miss extremely short-lived faults (a small number of packet losses, for example). It is also acceptable for the mechanisms to be unable to isolate faults long after they have occurred. We believe that a fault isolation mechanism cannot simultaneously achieve perfect *temporal fidelity*, scale and robustness (Section V).

In an earlier section we have argued that, by several measures, SNMP-based monitoring of every router in the network is insufficient for large-scale fault isolation. These arguments hold for monitoring multicast distribution trees as well. One pertinent criticism against SNMP-based monitoring was that routers lack the information necessary to identify application sessions that might be affected by a fault. That is, 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 maintain some *history* of the results of these probes. Once a fault is detected, one or more affected receivers may use this history information to isolate the fault. Before describing the probe history collection and fault isolation mechanisms in detail, we first describe the probing primitive that forms the basis for our fault isolation scheme.

### B. The Multicast Traceroute

Multicast routers support a multicast traceroute [56] primitive that is used for debugging Mbone connectivity problems. Multicast traceroute is a bit more sophisticated than its unicast counterpart. 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 (Fig. 3). 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, and a count of the total number of multicast packets forwarded down the tree. These counts help determine how many packets were dropped by this router. Second, the router forwards the request to the previous hop toward the source (the identity of the previous hop is determined from routing tables). This router, and each successive

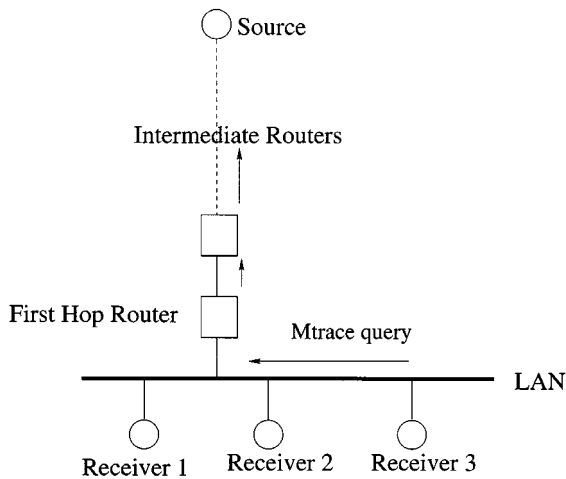


Fig. 3. Multicast traceroute. The multicast traceroute query is propagated hop by hop toward the source. Note that the “first hop router” in the figure is the last router on path from the source to the receivers, and is the router seen at the first hop of the mtrace query.

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. In this way, the receiver can determine both its path to the source, and the loss rates on individual links in that path.

The multicast traceroute protocol can be extended to *subcast* the mtrace response down one subtree of the distribution tree. Such a feature could be achieved quite easily by extending the semantics of the current multicast traceroute specification [56]. Mtrace requests already have a hop-limit field; this field is decremented at every router that the request traverses. The router at which this field would be decremented to zero can send the partial response to the request. An mtrace request can also direct the corresponding response to be multicast to a particular group. Subcast can be achieved by modifying the specification to send the response to a hop-limited request only down the subtree rooted at the node generating the request [Fig. 4].<sup>4</sup>

### C. Session Watchers and Session Watcher Coordination

In this subsection, we show how the multicast traceroute can be used to scalably monitor multicast distribution trees. The next subsection describes how to use the collected information to isolate faults.

1) *A Naive Approach*: There exists a naive technique for monitoring multicast distribution trees that uses multicast traceroute. Assume that there exists a software entity called a *session watcher* on each LAN that contains at least one receiver. 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 observed at the receiver. Locating the router responsible for a route change is harder:  $W_a$  needs to query neighboring session watchers to determine

<sup>4</sup>The ability to subcast mtrace responses is distinct from the more general capability of subcasting multicast packets [10]. This latter capability is not yet available in routers in the Internet. Such a feature could also, of course, be used to subcast mtrace responses.

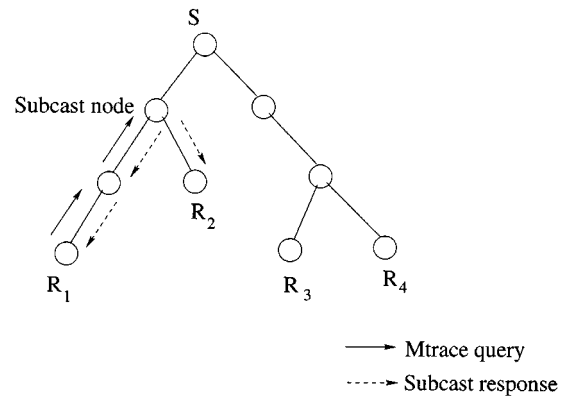


Fig. 4. Mtrace features. This figure illustrates subcasting the mtrace responses used in this paper to achieve scalable session watcher coordination.

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, 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 naive technique has some interesting properties. Other receivers 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. Furthermore, this technique scales better to large groups than a centralized collection of traceroutes (using, for example, third-party 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. We now describe a technique for collecting multicast traceroute responses that overcomes some of these deficiencies.

2) *Session Watcher Coordination*: Our technique 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 (Subsection B)—at the common ancestor between  $W_a$  and  $W_b$  (Fig. 5).

Of course, the previous paragraph omits several interesting details. 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 must trace its path once to the source. Further,  $W_a$  must subcast its mtrace response. Using this response,  $W_b$  can determine the answers to both these questions. In our example, 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 (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

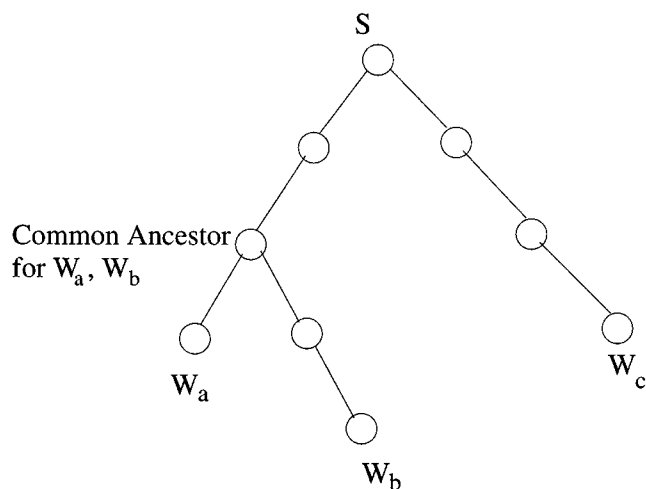


Fig. 5. Common ancestor. The session watcher  $W_b$  only needs monitors up to its common ancestor with  $W_a$  because  $W_a$  monitors the rest of its path to the source.

small number of subcast mtrace responses from  $W_a$ , it extends its mtrace request all the way to the source. 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 multicast traceroutes may overlap. Once  $W_b$  hears  $W_a$  subcasts again, it *backs off* its mtrace requests to the common ancestor again.

We have described our distribution tree monitoring technique with respect to two receivers (or session watchers). As we show in the next section, this technique generalizes to many receivers. However, even with this description, we see that our technique utilizes two of the design principles outlined earlier. The periodic subcasts of mtrace responses are a manifestation of *announce-listen*. They serve to inform other session watchers of the liveness of their originator. Furthermore, the subcasts also promote *shared-learning*. Using these, a session watcher can determine the common ancestor at which it can terminate its mtrace request.

Because of the ambiguity of interpretation to lack of feedback, our technique can cause a session watcher to adapt inappropriately. For example, a session watcher that fails to hear subcasts from other session watchers (caused by loss at a congested link) may *increase* its hop-limit and thereby add traffic to the congested link. There exist techniques, not described in this paper, that ensure that on average at most one session watcher responds inappropriately, thereby limiting the adverse effects of incorrect adaptation.

3) *Protocol Description*: In the following paragraphs, we describe our technique for monitoring multicast distribution trees. We do this by describing the sequence of actions executed at every session watcher  $W_a$ . Before describing these actions, we define some notation.

- 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  $T$ . This timer defines the periodicity of sending mtrace requests.

- $W_a$  maintains an *ancestor list*. This list, sorted by the number of hops to an ancestor, contains all ancestors that  $W_a$  shares with any other session watcher  $W_b$ .
- $W_a$  also sets a *backoff timer*. The duration of this timer is a small 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  $\infty$ . It sets the ancestor list to NULL, and starts the probe timer.

*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 (an  $\infty$  hop limit indicates that the request reaches the source).  $W_a$  also ensures that the response is multicast to a well-known group address to which all session watchers belong. In this manner,  $W_a$ 's mtrace request is subcast from a router that is at a distance indicated by the current hop limit.

*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  $\infty$  and clears out its ancestor list.

*Receiving another watcher's response*: When  $W_a$  receives a response to an mtrace request sent by  $W_b$ , then we have the following.

- 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  $d_{ab}^a$ ) and  $W_b$ 's distance to  $r_{ab}$  (call this  $d_{ab}^b$ ).
- 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  $d_{ax}^a$  such that  $r_{ax}$  is the nearest ancestor for which  $d_{ax}^a > d_{ax}^x$  (ties are broken by lower IP address). If there is no such ancestor, it sets its current hop limit to  $\infty$ .
- 4) If  $r_{ax}$  is the same as  $r_{ab}$ ,  $W_a$  starts a new backoff timer.

Intuitively, these steps allow  $W_a$  to back off its mtrace requests to the nearest ancestor to which some other watcher is closer to than  $W_a$ .

*Backoff timer expiration*: Suppose that, when the backoff timer expires,  $W_a$  is terminating its mtrace request at ancestor  $r_{ab}$ . From Step 4 in the previous paragraph, 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 according to Step 3 in the previous paragraph. Finally,  $W_a$  restarts the backoff timer.

We have simulated this session watcher coordination protocol in the network simulator *ns*. Reference [49] contains illustrations of how the protocol adapts to multicast tree dynamics such as receiver leaves, joins, route changes, as well as losses on the tree.

4) *Protocol Details*: Our description of the session watcher coordination protocol has omitted several details or otherwise

simplified the description of the protocol. We briefly address these here.

Every session watcher initially—and when a path change happens—sends an mtrace request all the way to the source. It does this in order to determine path overlap with other watchers, whose mtrace responses it receives. This approach clearly doesn't scale to large groups. An alternative approach would have the session watcher  $W_a$  start with an initial hop limit of 1.  $W_a$  would slowly increase the hop-limit until its current path fragment impinges on another session watcher's path.

The load on a router is proportional to the fanout of the multicast tree at that router. Thus, if there are  $N$  tree links incident on a router,  $N - 2$  mtrace requests will terminate at that router. In practice, we do not expect that processing these multicast traceroute messages will add significantly to router overhead.

In steady state (i.e., in the absence of any route changes or packet losses), any link in the multicast tree sees exactly one mtrace request propagating toward the source. However, because mtrace responses are subcast, a single link may see several mtrace responses. In particular, the number of mtrace responses traversing any link on the multicast distribution tree is the sum of the number of branches at each router on the path from that link to the source. For a binary tree, this number is bounded by the length of the path from the session watcher to the source. More generally, however, this subcast overhead may represent a scaling problem. We are investigating several approaches, one of which is adaptively adjusting  $T$  so that the long-term data-rate attributable to responses remains constant or increases slowly. Of course, this approach can result in reduced temporal fidelity of fault isolation.

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 corresponding receivers will clearly lack a fault isolation capability. Alternate techniques, such as MINC (Section IV), might be applicable in such cases.

#### D. Fault Isolation Using Session Watchers

The previous section has described a technique for scalably monitoring a multicast distribution tree. This technique has one key property: at any instant, a session watcher  $W_a$  knows all other watchers whose paths to the source overlap with its own (we use  $\mathcal{W}_a$  to denote this set). This property is very useful for isolating a fault. We describe how a session watcher may 1) locate a congested link in its path to the source, and 2) may approximately locate the origin of an observed route change.

To isolate a congested 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 pinpoint the bottleneck link on the path from  $W_a$  to the source. Thus, our session watcher coordination protocol isolates bottleneck 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  $\mathcal{W}$  before a route change.

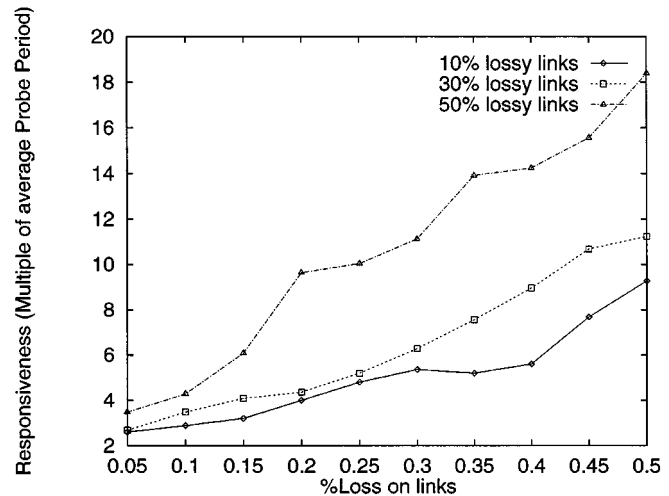


Fig. 6. Responsiveness versus loss rate.

Then, we assert that the router that originated the route change must lie between the lowest  $r_{ax}$  in the tree for which  $W_x$ 's path to the source has not been affected by the route change, and the first branch point in the tree on the path from  $r_{ax}$  to  $W_a$ . (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  $\mathcal{W}'_a$ . This number is bounded by the number of branch points on the path from  $W_a$  to the source. Furthermore, our tree monitoring scheme 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,  $W_a$  at least knows the identities of these routers and can use other techniques (e.g., SNMP) to more precisely locate the fault.

#### E. Protocol Performance

We have simulated our multicast monitoring scheme also in ns [1]. Fig. 1 described the behavior of our algorithm on a set of randomly generated tree topologies. These tree topologies were of size 60 nodes. Each node in the tree has an average fanout of 3. The generated trees had on an average 30 leaves (receivers). On these trees we randomly selected the lossy links and varied the percentage of losses. The duration of each simulation was 150 units.

The  $x$  axis on Fig. 6 plots the percentage of losses on the lossy links. The  $y$  axis shows the responsiveness which is the maximum time for which a node in the topology was unmonitored. Responsiveness is measured in multiples of the inter probe period. So, a responsiveness value of 3 means no node was unmonitored for 3 consecutive probe intervals. We obtained three different graphs by varying the number of lossy links in the topology.

Responsiveness is at least 2 in the presence of losses. As expected, responsiveness increases with increase in loss percentages as well as the number of lossy links in the topology. However, it is interesting to note that even when 30% of the links are lossy, each having 30% losses, every node in the topology is monitored at least once in six probe periods. We are currently

working on more extensive simulations to evaluate the overhead, adaptation, and accuracy of the monitoring scheme.

#### IV. RELATED WORK

Our distributed mapping scheme is an instance of a self-organizing distributed system. Early work in this area occurred in the context of packet radio networks. In these networks, mobile hosts collectively self-organize into *clusters* and establish intra-cluster topologies and transmission schedules [3], [12], [54]. Superficially, clusters are similar to ranges in our coverage protocol. As in our protocol, each station broadcasts the identities of stations it has heard from so that disjoint clusters are appropriately formed. However, range self-configuration differs from cluster formation in packet radio networks; range sizes change in response to failed surveyors or network partitions.

More recent research has proposed self-configuring techniques for 1) organizing application components into clusters with the goal of localizing interactions [36], [6], or 2) forming hierarchies of application components for scalable global interactions [52], [7]. As such, these represent different uses of some of the techniques described for robust self-configuration.

The literature on distributed architectures for configuration, fault detection, performance evaluation, accounting, and security management of networks is large [43], [16], [26], [44], [20], [28]. Our work represents an early exploration in trying to apply the robustness techniques to the problems that this research is attempting to solve.

Several tools already exist for topology discovery in campus-area networks. Some, such as *Fremont* [11], use a variety of sources of information (such as DNS, RIP, ARP, and ICMP) to extract router adjacencies. Others, such as *tkined* [29] and *InterMapper* [13], rely on SNMP interfaces. These tools can each be used to discover local topology in our surveyors (Section II). Tools with a larger scope, like *MapNet* [31], infer approximate but highly granular interprovider connectivity information from Internet backbone routing tables.

Commercially available network management systems provide decentralized, but localized, monitoring capabilities. In these systems, one *management agent* oversees a set of LAN's or devices. These agents are in turn polled by a central management platform. These systems are used for fault diagnosis, performance measurement, and traffic characterization. Examples of such systems include IBM's System View/Netview [25], HP's Openview [23], SUN's Solstice [53], and the Cabletron Spectrum [4]. All these platforms use RMON, an SNMP standard for the monitoring of segments such as Ethernet, token ring networks, switches, or hubs. Our surveyor will use similar techniques to gather strands from within their ranges.

Our approach to fault isolation using periodic end-to-end probing draws inspiration from recent work on network performance measurement. One approach, *NIMI* [27], proposes to establish a distributed network measurement infrastructure. Measurements are collected by probing paths between select nodes in the network [45]. Another proposed approach uses unicast probing and linear decomposition techniques to compute the performance of network elements.

Of most relevance to our work on fault isolation is work that infers bottleneck links and the multicast logical tree topology by correlating packet losses observed at receivers. Such inference is either based on rigorous statistical techniques [5] or on heuristics [48]. 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, they require centralized correlation of receiver loss patterns.

Multicast traceroute has been used in protocols for reliable multicast to determine tree topologies [34], [35]. In these schemes, receivers use multicast traceroutes to discover the nearest peer on the other side of a bottleneck link.

The MHealth [37] tool is similar to our approach in that it also uses multicast traceroute to monitor multicast groups. The MHealth tool, however, operates from a single location and collects real time protocol (RTP) [51] updates and also queries paths between various receivers and sources of a multicast group using multicast traceroute. Losses and tree topology thus discovered are visually presented. The paper expresses concerns about the scalability of this approach when multiple MHealth tools are initiated to monitor the same session. This is because of the uncontrolled multicast traceroute traffic that can be potentially generated by the various MHealth tools monitoring the same session.

Of relevance to our work are techniques for proactive fault isolation. One such approach [24] *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 session watchers as well.

#### V. CONCLUSION

In this paper, we have described two protocols that represent a novel approach to designing tools for fault isolation. This approach emphasizes careful distribution of data collection for robustness and scale, with a possible reduction of monitoring fidelity. We believe these tradeoffs—having possibly slightly incomplete maps under some pathological conditions, ignoring very short-lived faults, or only localizing the fault to some small topological neighborhood—are acceptable.

We are currently pursuing some future directions, especially with respect to the multicast tree fault isolation work. First, our distribution tree monitoring infrastructure can scale better by sharing path information across sessions. For example, a session watcher may not need to probe path segments of two different trees (corresponding to different sources of the same multicast group, or receivers on the same LAN belonging to different groups) if there is significant overlap on the path. Second, our tree monitoring procedures may themselves be informed by application traffic patterns. For example, rather than sending mtrace requests periodically, session watchers may trigger an

mtrace request after some number of observed losses in an audio stream.

#### ACKNOWLEDGMENT

The authors would like to thank V. Paxson (ACIRI) for the extensive comments on earlier versions of this draft. The authors would also like to thank M. Handley (ACIRI) who also contributed to the ideas presented in this paper.

#### REFERENCES

- [1] S. Bajaj, L. Breslau, and D. Estrin *et al.*, "Improving simulation for network research," Univ. Southern Calif., Los Angeles, Tech. Rep. 99-702, Mar. 1999.
- [2] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The Harvest information discovery and access system," *Computer Networks ISDN Syst.*, vol. 28, pp. 119–125, 1995.
- [3] C. V. Ramamoorthy, J. Srivastava, and W.-T. Tsai, "Clustering techniques for large distributed systems," in *Proc. IEEE INFOCOM*, 1986.
- [4] Cabletron Systems. Spectrum. [Online]. Available: <http://www.cabletron.com/spectrum/>.
- [5] R. Caceres, N. G. Duffield, J. Horowitz, and D. Towsley, "Multicast-based inference of network-internal loss characteristics," *IEEE Trans. Inform. Theory*, vol. 45, Nov. 1999.
- [6] C. Bormann, J. Ott, and N. Seifert, "Mtp/so: Self-organising multicast," Internet Draft, May 1997.
- [7] C. Alaettinoglu, D. Estrin, S. Kumar, and D. Thaler. Self-configuring hierarchy for interdomain pim. [Online]. Available: <http://www.isi.edu/kkumar/hpim>.
- [8] K. M. Chandy, A. Rifkin, and E. Schooler, "Using announce-listen with global events to develop distributed control systems," in *Proc. ACM Workshop High Performance Java Network Computing*, Palo Alto, CA, Feb. 1998.
- [9] Cisco Systems. Netflow. [Online]. Available: <http://www.cisco.com/warp/public/732/netflow/index.html>.
- [10] A. Costello and S. McCanne, "Search party: Using randomcast for reliable multicast and local recovery," in *Proc. IEEE INFOCOM*, New York, Mar. 1999.
- [11] D. C. M. Wood, S. S. Coleman, and M. F. Schwartz, "Fremont: A system for discovering network characteristics and problems," in *Proc. USENIX Winter Conf.*, Jan. 1993, pp. 223–347.
- [12] D. J. Baker, "Distributed control of broadcast radio networks with changing topologies," in *Proc. IEEE INFOCOM*, 1983.
- [13] Dartmouth Univ. InterMapper: An intranet mapping and snmp monitoring program for the Macintosh. [Online]. Available: <http://www.dartmouth.edu/netsoftware/intermapper>.
- [14] S. Deering, "Multicast routing in internetworks and extended LAN's," in *Proc. 1988 ACM SIGCOMM Conf. Commun. Architectures Protocols*, Aug. 1988, pp. 55–64.
- [15] R. C. F. Baker, "OSPF version 2 management information base," Internic Directory Services, Request for Comments 1248, July 1991.
- [16] F. Stamatelopoulos, T. Chiotis, and B. Maglaris, "A scaleable, platform-based architecture for multiple domain network management," in *IEEE Int. Conf. Commun.*, June 1995.
- [17] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397–413, Aug. 1993.
- [18] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proc. ACM SIGCOMM '95*, Aug. 1995.
- [19] F. Baker and G. Malkin, "RIP version 2 MIB extension," Internic Directory Services, Request for Comments 1724, Nov. 1994.
- [20] G. Goldszmidt and Y. Yemini, "Distributed management by delegation," in *Proc. 15th Int. Conf. Distributed Computing Syst.*, June 1995.
- [21] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.
- [22] M. Handley and J. Crowcroft, "Network text editor (NTE): A scalable shared text editor for the MBone," in *Proc. ACM SIGCOMM '97 Computer Communication Rev.*, vol. 27, Oct. 1997, pp. 197–208.
- [23] Hewlett Packard. Hp Openview. [Online]. Available: <http://www.hp.com/openview/index.html>.
- [24] C. S. Hood and C. Ji, "Proactive network fault detection," in *Proc. IEEE INFOCOM*, Kobe, Japan, Apr. 1997.
- [25] IBM. Netview for aix. [Online]. Available: <http://www.networking.ibm.com/nv6/nv6prod.html>.
- [26] International Telecommunication Union, "Principle and architecture for the tmn," Geneva, Recommendation M.30310, 1992.
- [27] J. Mahdavi, M. Mathis, and V. Paxson. (1997, May) Creating a national measurement infrastructure (nimi). [Online]. Available: <http://www.psc.edu/networking/nimi>.
- [28] J. Schonwalder, "Using multicast-snmpp to coordinate distributed management agents," in *2nd Int. Workshop Syst. Manage.*, Toronto, Canada, June 1996.
- [29] J. Schonwalder and H. Langendorfer, "How to keep track of your network configuration," in *Proc. 7th Conf. Large Installation Syst. Admin. (LISA VII)*, Monterey, CA, Nov. 1993, pp. 189–193.
- [30] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM Symp. Commun. Architectures Protocols*, Stanford, CA, Aug. 1988.
- [31] K. Claffy and B. Huffaker. Macroscopic internet visualization and measurement. [Online]. Available: <http://www.caida.org/Tools/Mapnet/summary.html>.
- [32] S. Kalindi and M. J. Zekauskas, "Surveyor: An infrastructure for Internet performance measurements," in *Proc. INET*, San Jose, CA, June 1999.
- [33] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing instability," in *Proc. ACM SIGCOMM '97*, Sept. 1997.
- [34] B. N. Levine, S. Paul, and J. J. Garcia-Luna-Aceves, "Organizing multicast receivers deterministically according to packet-loss correlation," in *Proc. 6th ACM Int. Multimedia Conf. (ACM Multimedia 98)*, Sept. 1998.
- [35] D. Li and D. R. Cheriton, "Oters (on-tree efficient recovery using sub-casting)—A reliable multicast protocol," in *Proc. 6th IEEE Int. Conf. Network Protocols (IEEE ICNP '98)*, Oct. 1998.
- [36] L. Zhang, S. Floyd, and V. Jacobson, "Adaptive web caching," White Paper, Feb. 1997.
- [37] D. Makofske and K. Almeroth, "MHealth: A real-time multicast tree visualization and monitoring tool," in *Proc. Network Operating Syst. Support Digital Audio Video (NOSSDAV)*, Basking Ridge, NJ, June 1999.
- [38] D. Massey and B. Fenner, "Fault detection in routing protocols," in *7th Int. Conf. Network Protocols*, Toronto, Canada, Nov. 1999.
- [39] S. McCanne and V. Jacobson, "Vic: A flexible framework for packet video," in *Proc. ACM Multimedia '95*.
- [40] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driver layered multicast," in *Proc. ACM SIGCOMM 1996*, Stanford, CA, Aug. 1996, pp. 117–130.
- [41] S. R. McCanne, "Scalable multimedia communication with internet multicast, light-weight sessions, and the mbone," Computer Science Division, Univ. of California, Berkeley, Tech. Rep., 1998.
- [42] K. McCloghrie and M. Rose, Management Information Base for Network Management of TCP/IP-based internets: MIB-II, Internic Directory Services, Mar. 1991.
- [43] M. Kahani and H. W. P. Beadle, "Decentralised approaches for network management," *ACM SIGCOMM Computer Commun. Rev.*, vol. 27, no. 3, July 1997.
- [44] P. Alexander and K. Carpenter, "Atm net management: A status report," *Data Commun. Mag.*, 1995.
- [45] V. Paxson, "End-to-end routing behavior in the internet," in *Proc. ACM SIGCOMM Symp. Comm. Architectures Protocols*, San Francisco, CA, Sept. 1996.
- [46] —, "End-to-end Internet packet dynamics," in *Proc. 1997 ACM SIGCOMM Conf. Commun. Architectures Protocols*, Sept. 1997.
- [47] R. Govindan, C. Alaettinoglu, and D. Estrin. Self-configuring active network monitoring (scan). [Online]. Available: [http://www.isi.edu/scan/Pubs/scan\\_proposal.ps.gz](http://www.isi.edu/scan/Pubs/scan_proposal.ps.gz).
- [48] S. Ratnaswamy and S. McCanne, "Inference of multicast routing tree topologies and bottleneck bandwidths," in *Proc. IEEE INFOCOM*, NY, Apr. 1999.
- [49] A. Reddy, D. Estrin, and R. Govindan, "Large-scale fault isolation," Univ. Southern Calif., Los Angeles, Tech. Rep. 99-706, 1999.
- [50] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet," in *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999.
- [51] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A transport protocol for real-time applications, Internic Directory Services, Jan. 1996.

- [52] P. Sharma, D. Estrin, S. Floyd, and L. Zhang, Scalable session messages in SRM, submitted for publication.
- [53] SUN Microsystems. Solstice. [Online]. Available: <http://www.sun.com/solstice/>.
- [54] T. G. Robertazzi and P. E. Sardchik, "Self organizing communication networks," *IEEE Comm. Mag.*, vol. 24, pp. 28–33, Jan. 1986.
- [55] T. Turletti, "INRIA Videoconferencing System (ivs)," *ConneXions* 8.
- [56] W. Fenner and S. Casner, "A traceroute facility for ip multicast," Internet draft, Aug. 1998.
- [57] D. Waitzman, C. Partridge, and S. E. Deering, "Distance vector multicast routing protocol," Internic Directory Services, Request for Comments 1075, Nov. 1988.
- [58] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOMM*, San Francisco, CA, Sept. 1996, pp. 40–52.

**Anoop Reddy** received the B.Tech. degree in electrical engineering from the Indian Institute of Technology at Mumbai, India, in 1994.

He is a Doctoral student in computer science at the University of Southern California, and a Research Assistant at the University of Southern California's Information Sciences Institute. His research interests include internet routing, network measurements, and monitoring for performance analysis and fault diagnosis.

**Deborah Estrin** received the Ph.D. and M.S. degrees from the Massachusetts Institute of Technology in 1985 and 1982, respectively, and the B.S. degree from the University of California at Berkeley in 1980.

She is a Professor of computer science at the University of Southern California in Los Angeles, where she joined the faculty in 1986. She is a co-PI on the DARPA Virtual Internet Testbed (VINT) project, the DARPA Scalable Coordination Architectures for Deeply Distributed Systems (SCADDS) project, and the NSF Routing Arbiter project at USC's Information Sciences Institute, where she spends much of her time supervising doctoral student research. While she continues her research related to protocol scaling and multicast, most recently she has begun to focus on problems related to networking and coordination among very large numbers of physically embedded devices (sensors, actuators). She is an active member of the IETF multicast routing related working groups and a long-time member of the End-to-End research group.

Dr. Estrin is a member of the ACM and AAAAS. She has served on numerous panels for the National Science Foundation, the National Academy of Engineering, and DARPA's Information Systems and Technology (ISAT) advisory board. She has served as an Editor for the *ACM/IEEE TRANSACTION ON NETWORKS* and the *Wiley Journal of Internetworking Research and Experience*, and as a member of the program committee for many IEEE Infocom and ACM Sigcomm conferences. She was program Co-chair of ACM Sigcomm 1996. She received the National Science Foundation, Presidential Young Investigator Award for her research in network interconnection and security.

**Ramesh Govindan** received the Ph.D. and M.S. degrees in computer science from the University of California at Berkeley, in 1992 and 1989, respectively, and the B.Tech. degree in computer science and engineering from the Indian Institute of Technology at Madras, India, in 1987.

He is Project Leader at the University of Southern California's Information Sciences Institute and a Research Assistant Professor of computer science at the University of Southern California. He also worked for two years at Bell Communications Research, Morristown, NJ. His research interests include internet unicast and multicast routing, and networking numerous small devices.