Real-Time Multicast Tree Visualization and Monitoring[*]

David B. Makofske
Dept of Computer Science
University of California
Santa Barbara, CA 93106-5110
davidm@cs.ucsb.edu

Kevin C. Almeroth
Dept of Computer Science
University of California
Santa Barbara, CA 93106-5110
almeroth@cs.ucsb.edu

March 24, 2000

Abstract

The exponential growth of the Internet combined with the increasing popularity of streaming audio and video are pushing Internet bandwidth constraints to their limits. Methods of managing and more efficiently utilizing existing bandwidth are becoming increasingly vital. Using IP multicast to deliver content, especially streaming audio and video, can provide enormous bandwidth savings. A decade of effort at deploying multicast, combined with the rising need for better traffic management for bandwidth-hungry applications has led to significant momentum for multicast use and deployment. One of the remaining barriers to widespread adoption is the lack of multicast monitoring and debugging tools. To address this need we introduce *MHealth*, a graphical, near real-time multicast monitoring tool. *MHealth* utilizes existing tools to collect comprehensive data about Realtime Transport Protocol (RTP) based streaming audio/video sessions. By using a combination of application-level protocol data for participant information and a multicast route tracing tool for topology information, *MHealth* is able to present a multicast tree's topology and information about the quality of received data. In this paper we describe the design and implementation of *MHealth* and include an example analysis of multicast tree statistics.

Keywords: multicast, management, monitoring, topology, routing, tree determination

[*]This work has been supported in part by a research grant from Cisco Systems Inc. through the University of California MICRO Program.

1 Introduction

Exponential growth in the Internet continues to dramatically increase the total number of connected people. In addition to the shear volume of new users, the amount of the data transfered has grown as more applications use innovative media like hypertext, pictures, audio, video, etc. One of the most significant examples of applications that are more bandwidth-intensive are those that use streaming media. The result is that an increase in users and data has produced considerable strain on the existing Internet infrastructure. This has led to a number of highly publicized network and server failures; particularly when major events such as political scandals, international crises, or entertainment events have attracted users to request streaming audio/video content en mass. The growth trends suggest these events, and the congestion they cause, will increase in scope and frequency.

Wide area bandwidth is clearly not keeping up with customer demand. Furthermore, the ability of end stations to send/receive increasing volumes of data will only increase. A main contributer is the rapidly increasing bandwidth capacities in the "last mile"–one of the few remaining bottlenecks helping to restrict user data flow into and out of the Internet. As computers become faster, and cable modems and digital subscriber lines (DSL) become more common, Internet congestion will only become worse. Methods must be implemented to manage and utilize the existing Internet bandwidth more efficiently. These methods can take many forms, such as moving the content closer to the requesters (caching and replication), more conservative sending techniques (request aggregation), and increasing the functionality provided by the routers (multicast and quality of service techniques).

Multicast offers a compelling bandwidth management technique for streaming audio and video. This is especially true when it is used for broadcast-style applications, e.g. when a pre-recorded or live multimedia stream is sent to many receivers simultaneously. In contrast to unicast, which sends from a single source to a single destination, multicast allows a source to send data once and have it reach a group of interested receivers. The source sends packets to a multicast group address and the network routers replicate the packets when the paths to the receivers diverge. Each multicast-enabled router ensures that packets flow on the correct links to reach all of the receivers who have joined a particular multicast group. When a streaming media broadcast is sent with unicast, there is a high redundancy of data transfer on each network link, but with multicast, a single stream is likely to never traverse the same network link more than once. With high quality MPEG[1] at several Mbps per stream, multicast would greatly reduce load on the network and on the server. Depending on the location and number of receivers, the bandwidth saved with multicast can be many orders of magnitude.

Despite the conceptual simplicity of multicast and its obvious benefits, its implementation is difficult. After much research experimentation, multicast was initially deployed as the research-oriented Multicast Backbone (MBone)[2]. has now grown beyond a research network and is beginning to be deployed by Internet Service Providers (ISPs) and used by service-oriented companies[3]. However, the growing popularity of real-time audio and video traffic and the aforementioned network failures have created a compelling business justification for multicast. Organizations such as the IP Multicast Initiative (IPMI)[1] have been working to educate consumers about the benefits of multicast, and major ISPs have been implementing and promoting multicast as a network service. Broadcast.com (now Yahoo!), a leading provider of streaming media content on the Web, has been actively encouraging the use of multicast. Furthermore, commercial tools such as Real Network's RealPlayer and Microsoft's NetShow are now multicast enabled. Almost all router manufacturers now include some multicast capability as a standard feature on their routers.

Many ISPs are still reluctant to enable multicast on their networks because of the difficulty of properly managing multicast[4]. Deployment of multicast is not a simple of matter of configuring routers and switches but also requires significant expertise and management capabilities. There are few tools available for monitoring and debugging multicast networks, and the tools that do exist are difficult to use. Furthermore, the philosophy in developing multicast tools has been to try and duplicate the functionality of unicast tools. While this makes the tools easier to understand, it limits their utility. There is a strong need for multicast debugging and monitoring tools that are graphical, intuitive to use, and display multicast as a one-to-many network service. While some multicast monitoring tools exist, none meet our goals of working in today's Internet and providing both tree visualization and reception feedback in near real-time. As more and more companies and ISPs activate multicast in their networks, this will become an increasingly critical need.

This paper describes *MHealth*, the Multicast Health Monitor, which is a graphical, near real-time multicast monitoring tool. *MHealth* handles the unique characteristics of multicast traffic by collecting a comprehensive set of data about a session. By using a combination of application level protocol data about group participants, and a multicast route tracing tool for topology information, *MHealth* is able to discover and display the full multicast distribution tree and delivery quality. *MHealth* also provides data logging functionality for the purpose of off-line analysis. As an example of this function, we present an analysis of data collected using *MHealth*. Finally, while the concept of *MHealth* has proven successful, the technology underlying the tool could be improved. We discuss some of the limitations of *MHealth* and propose ways of making improvements.

The remainder of this paper is organized as follows. Section 2 discusses related work in the field of multicast network monitoring and multicast data analysis. Section 3 introduces the *MHealth*

[1]http://www.ipmulticast.com/

tool, discussing its design, implementation, and some observational analysis. Section 4 evaluates the effectiveness and issues of *MHealth*, and section 5 presents future work. Section 6 summarizes the work and presents conclusions.

2 Related Work

There are a number of existing tools for monitoring, debugging, and analyzing multicast traffic and data flow[4]. These tools have mostly evolved out of a need to debug connectivity problems and multicast routing bugs. As multicast has evolved into more of a commercial service, these tools have struggled to fill the needs of network administrators. Today, tools are needed which are more suited for identifying whether multicast traffic and trees are "healthy", i.e. whether a group's traffic is of acceptable quality, whether traffic is reaching all of the group's members, and whether traffic is flowing across unnecessary links.

One of the most important tools in use today is *mtrace*[5], a multicast version of *traceroute*. *Mtrace* works by starting at one of a group's receivers and tracing the *reverse* path back to the source. The reverse path is used since this is how group members are added to the group. Receivers join a group by sending a join towards the source. This join messages travels a reverse path, "reverse" in the sense that it is opposite of how the traffic will flow. As this join message travels towards the source, forwarding state is created. This allows the source to send packets over the tree created by the forwarding state. This also means that the source can send packets to a group without knowing who the members are. *Mtrace* is one of the primary data sources used by *MHealth* and will be discussed in detail later in the paper.

Several tools provide statistics based on the Realtime Transport Protocol (RTP) and its associated control protocol the Realtime Transport Control Protocol (RTCP)[6]. Many of the multicast tools use RTCP to create a list of group members. *RTPmon*[7] collects and displays this information along with real-time statistics about packet loss and jitter. *Mlisten*[8] uses RTCP packets to collect and archive group statistics for all MBone groups advertised through *sdr*. MultiMON[9] monitors the multicast traffic on a local network segment and provides graphs on media types and amounts.

In addition to these tools, there are several SNMP-based management tools. Merit Network has developed a suite of tools including: *mstat*, *mrtree*, and *mview*[10]. *Mstat* allows an SNMP-enabled router to be queried for information, including routing tables and packet statistics. *Mrtree* uses cascaded SNMP router queries to provide a text-based representation of a particular multicast group's topology. *Mview* is a tool for visualizing MBone topology as well as monitoring and collecting data from *mrinfo*, *mtrace*, *mstat* and *mrtree*. The topology construction function requires a user to click on a node and specify one or more information finding actions. These tools are well-suited for use within an administrative domain. They are not suited for the inter-domain because one ISP is unlikely to give detailed access to either end users or network managers in other networks.

Several papers have been published on the analysis and behavior of multicast groups in the MBone. However, these works have tended not to focus on real-time visualization. In work by Handley, tools were written to log RTP/RTCP packets and collect *mtraces* for an MBone session[11]. The data sets were then used to manually create a picture of the multicast tree and various statistics about links in the tree. Yajnik, et al. characterize a controlled MBone session by analyzing packet loss statistics from 11 participating sites[12]. The data is examined for spatial and temporal correlation. Almeroth and Ammar use *mlisten* to collect group membership data for all MBone sessions over an extended period of time[13]. Analysis efforts focused on modeling temporal, spatial, inter-session and intra-session characteristics. Our goal with *MHealth* is to provide some of these group characteristics in real-time while also logging the data for later analysis.

3 The *MHealth* Tool

In this section we present the *MHealth* tool, a prototype developed with the goal of exploring the difficult task of real-time multicast tree estimation. Our description of *MHealth* includes the design of the tool including what underlying tools and data are required; a description of how tree data is presented and how a user can use the tool. We also provide a sample data set collected with *MHealth* and show sample analysis of how the data can be used to provide insight into tree characteristics.

3.1 Design

An understanding of *MHealth* operation is such that it necessitates some background on how streaming audio and video are sent on a multicast channel. Since multicast transmission for real-time streaming applications is one-to-many, the Transport Control Protocol (TCP) is not a suitable. Every TCP packet requires an acknowledgment, and requiring many receivers to acknowledge every packet would soon overwhelm the source with acknowledgments. This situation is referred to as *ACK implosion*. In order to scale to large number of receivers, IP multicast usually uses the User Datagram Protocol (UDP) as its transport protocol. UDP provides a connectionless service and does not guarantee against lost, duplicated, or out-of-order packets. This leaves the issue of how to send real-time data over multicast without all of the services of TCP; most importantly, reliability and congestion control. Furthermore, real-time data has very stringent requirements in terms of packet inter-arrival and delay. While data should be delivered as accurately as possible, the real-time nature of streaming media prohibits retransmission of lost packets. Finally, UDP's connectionless nature means that without connection-oriented endpoints for data flow, it is difficult to determine the receivers of a multicast stream and the routes the data must take to reach them.

In order to provide better support for streaming media, applications commonly use an application layer protocol, sometimes called Application Layer Framing (ALF)[14]. An ALF-style protocol can take the form of a separate protocol layer between the application and the transport layer, or it can be integrated into the application itself. ALF-style protocols add important functionality such as reordering and packet timing on top of UDP's port multiplexing services. One ALF-style protocol used for streaming real-time audio and video is RTP. It is used in the MBone tools as well as in several commercial streaming tools. RTP provides payload type identification, sequence numbering, and timestamping on top of UDP. RTCP, the control protocol for RTP, allows participants in a multicast group to report their membership and the quality of their reception. The protocol specification states that RTCP packets are to be sent by each group member on the group's multicast address but using a different port number.

Multicast monitoring tools are a more difficult task than TCP-based tools. Simply put, TCP-based tools can take advantage of state information that exists at network endpoints. No such state exists for UDP-based protocols like multicast. There is no transport level connection setup, and the source does not known who receiver(s) are. There are two basic mechanisms that can be used to collect multicast group information: query the routers for network-layer state, or query group members for application-layer state. For group members, RTCP is one example of a protocol used for exchanging application-layer state. To obtain network layer state, there are a number of options. In one case, routers can be queried directly. Queries are made using SNMP or by logging in to the router and dumping the router's state. However, frequent, direct access can be costly in terms of overhead and it can create security risks. Another option is to query routers using a less resource intensive and more controlled interface. Tools like *traceroute* and *ping* use this mechanism. Most existing management tools have relied on using either network-layer data or application-layer data but not both. In the few cases where tools do use both methods, they typically display the results separately and do not attempt to integrate them.

MHealth integrates both application layer data and in-direct router-based information obtained from *mtrace* into a single monitoring tool. *MHealth* relies on the application layer protocol information from RTCP packets to determine the group membership and the delivery quality (jitter, delay, and loss) at each participant. Once *MHealth* has established a group's participants, the *mtrace* utility is used to trace the hops in the tree. As a path between each source and receiver is established, it is combined into a tree data structure and graphically displayed. The details of these data sources are now described.

3.1.1 Real Time Control Protocol

RTCP is defined as part of the RTP standard. The RTP portion of the protocol applies application level framing for real-time data, providing payload type identification, sequence numbering, and timestamping. The RTCP portion of the protocol is a periodic transmission of control packets by all group members to all other group members. In the MBone, RTCP packets are usually sent on the same multicast address as the RTP data itself, but on a different UDP port. Typically the data port is an even number, n, and the control port is $n+1$. RTCP is described as having four primary functions.

1. RTCP provides feedback on the quality of the data transmitted to the multicast group. This is a critical transport function of RTP and can be used to develop and implement flow and congestion control, adaptive encoding techniques, and fault diagnostics.
2. RTCP carries a persistent transport-level identifier for an RTP source called the *canonical name*. It is used to synchronize data from multiple tools (such as audio and video).
3. RTCP packets are used by each participant to estimate the group size. This estimate is important for scaling the RTCP send rate of each group member. This function helps make RTCP scalable, and prevents members in large groups from causing congestion solely based on control traffic.
4. RTCP provides distribution of group membership information. This is meant to be used as an informal mechanism. However, it is not a reliable accounting mechanism.

Each RTCP packet consists of one or more packet sections. The key sections used by *MHealth* include the following: the *sender report*, the *receiver report*, the *source description*, and the *BYE* section. If the packet sender is an RTP source (as well as a receiver), it will include a *sender report* which contains information about the RTP data being sent. If the packet sender is receiving RTP data from one or more sources, they will send a *receiver report* per RTP data source, up to a maximum of 32 per RTCP packets. These reports contain reception statistics about each data source. The *source description* section contains one or more text elements, including the canonical name, which describes the RTCP sender. If the *BYE* section is present, it indicates the RTCP sender is leaving the multicast group. A *BYE* packet should be the last RTCP packet heard from a group member unless they later rejoin the group. Because a *BYE* packet could be lost, some mechanism must be used to eventually remove group members. A receiver will "time out" if no RTCP packet has been heard from the receiver for a reasonable period of time.

In order for RTCP to scale to potentially very large multicast sessions, the send rate for RTCP packets must be controlled. If the amount of RTCP traffic is not controlled, it could grow linearly with the number of group members. For large groups, this could swamp the group with control information. To avoid this problem, each group member uses an algorithm to determine an RTCP send interval. As the group size increases the time between RTCP transmissions for each member also increases. For small sessions, RTCP packets are usually sent by each group member once every 5 seconds. In very large groups, members may send RTCP packets only once every couple of minutes. RTCP's goal is to limit the control traffic bandwidth to less than 5% of RTP data bandwidth. Of interest to *MHealth* is the granularity at which RTCP packets are sent. Large groups do not send RTCP reports frequently and so it is difficult to monitor members in large groups with fine granularity. Dealing with large groups is addressed later in the paper.

The richness, granularity, and completeness of the data provided by RTCP make it a logical choice as a data source for multicast group monitoring and management. Few other mechanisms provide any kind of information similar to that provided by RTCP. The RTP specification strongly recommends that all participants send RTCP packets, and most implementations currently do. It is still possible that not all RTCP packets are received. Some of the reasons include firewalls, unidirectional links, UDP packet loss, or tools that do not conform to the RTP specification. However, RTCP packets are the only standardized way to determine group membership of a multicast session. Currently, RTCP is the best mechanism available for a tool like *MHealth* to ascertain group membership information. In the case where RTCP is not used, some other mechanism must be found to determine who group members are. This may not be an easy task but is necessary for many management applications, especially those like *MHealth*.

3.1.2 *Mtrace* Utility

Once participants in a multicast session are identified, the topology must be discovered. The *mtrace* utility, a multicast version of *traceroute*, can provide this information[5]. *Mtrace* also provides additional information such as total multicast packets per hop, group-specific hop counts, and packet loss per hop. One of the disadvantages of *mtrace* is that it only reports the route from one participant to the source at a time. So *MHealth* must perform numerous *mtraces*, one for each participant in a session. *MHealth* then uses these traces to build a tree topology. In tree visualization, *MHealth* provides links to the other information called by *mtrace*.

Although the output from *mtrace* is clearly analogous to the *traceroute* output, the underlying tracing mechanism has a significantly different design. A unicast *traceroute* sends a series of packets with increasing Time-to-Live (TTL) values. As the TTL expires at each hop, an Internet Control Message Protocol (ICMP)[15] packet is returned to the source. This ICMP packet identifies the router at which the TTL expired. By listing the series of routers that return ICMP messages, an estimate of the path from the source to the destination can be built.

The approach used by *traceroute* cannot be used in multicast because ICMP TTL expiration messages are explicitly suppressed. These messages are suppressed for multicast addresses because multicast relies heavily on TTL scoping to control flooding. This requires a different technique for determining the multicast path from a source to a receiver. The approach taken has been to require all multicast routers to have customized functionality to respond to multicast trace requests. While this increases the overhead and complexity of the router, it does provide crucial diagnostic information.

Multicast trees are constructed in the reverse direction–receivers initiate a join message which flows toward the source. As a result of this difference in how unicast and multicast determine traffic flow, the tracing mechanism for multicast is substantially different than the techniques used for tracing a unicast path. Each router in a multicast tree does not know which receivers it is sending to; it merely knows the incoming and outgoing interfaces that each distinct group and source pair should flow on. This state is maintained by the multicast routing protocol for intermediate hops[16, 17, 18], and by the Internet Group Management Protocol (IGMP)[19] for the last hop (leaf) router. At each router only the outgoing interfaces that traffic should flow on is known. Which of those outgoing paths a given receiver is on cannot be determined. As a result, it is not possible to use the unicast *traceroute* method of tracing from a source to a particular receiver. The solution is to start at the receiver's location, and travel backwards toward the source. Since each router knows the incoming interface of a group's data–it is the interface the router would use to get back to the source–a path can be determined hop-by-hop from the receiver to the source. This is called a *reverse path lookup.* The reverse path is also the same path the receiver's join message took. Therefore, it is the same path that will be used for the source's traffic.

The reverse path lookup requires that routers be able to process a special *mtrace* IGMP *Query* packet. This packet is multicast by an *mtrace* initiator on the ALL-ROUTERS multicast address (224.0.0.2). The last hop router for the receiver specified in the *Query* packet recognizes it is the last hop router and initiates the *mtrace.* The last-hop router appends its data to the *mtrace* packet; alters the packet type from *Query* to *Request*; and forwards the packet via unicast to the previous router (the incoming interface for the source-group pair being traced). This continues up the path to the source until it reaches the router directly connected to the source. This router realizes the source is directly connected to it; appends its own information to the packet; and alters the packet type from a *Request* to a *Response.* The final packet is then either sent via unicast or multicast (according to packet field settings) to the *mtrace* initiator. If a router along the reverse path is having trouble contacting the next upstream router, the *mtrace* initiator will be sent the trace data up to that point. If the previous hop router cannot be contacted within a period of time, the *mtrace* will timeout and fail.

The added functionality required by the router allows *mtrace* to be more flexible and informative than a normal *traceroute*. One of the most important differences between *mtrace* and *traceroute* is that *mtrace* allows third-party mtraces, i.e. the initiator need not be the source or the destination. The IGMP query request header includes the multicast group address, the source address, the destination (receiver) address, and the response address where the *mtrace* data should be returned. In addition to this flexibility, *mtrace* collects and returns more comprehensive information. Data returned includes (1) the total number of packets received and transmitted on an interface, (2) a group-specific count of incoming and outgoing packets (if a group is specified), (3) the multicast routing protocol used, (4) the TTL required to reach the particular router, and (5) the explicit error messages when an *mtrace* cannot complete. This data, from each router, is appended to the *Request* packet as it flows from the receiver to the source.

MHealth relies heavily on *mtrace* and the data it collects. A decision early in the prototyping effort led us to run an existing implementation of *mtrace* and then parse the results. We believed the complexity of implementing *mtrace* from scratch would have been excessive for a prototype. In hindsight, it would have been more efficient and offered better flexibility to have implemented *mtrace* inside of *MHealth*.

3.1.3 Mtracing Hierarchy

When *MHealth* was first run on large groups, there were a fairly significant number of *mtrace* failures. On further investigation, several problems were isolated which required different command line options to correct. In many cases, the sets of options required are orthogonal and cannot be combined into a single execution of an *mtrace*. By running multiple *mtraces* we can increase the percentage of *mtraces* that are successful.

Up to three different *mtraces* are attempted for each receiver before *MHealth* moves on to the next receiver. First, a standard *mtrace* from a receiver to the group source is attempted. If it fails, a *gateway mtrace* is attempted. One of the major reasons why an initial *mtrace* fails is because the *Query* cannot find the receiver's last hop router. A gateway *mtrace* solves this problem by explicitly contacting the last hop router via its unicast IP address rather than attempting to multicast the *mtrace* request. The challenge is first to determine the last hop router. This is accomplished by doing an *mtrace* from the machine running *MHealth* to the receiver. This *discovery mtrace*, if successful tells *MHealth* the IP address of the last hop router. Then, a gateway *mtrace* is run.

Loss	**Router Color**	**End Host Color**
$loss < 2\%$	white	green
$2\% < loss < 10\%$	yellow	yellow
$loss \geq 10\%$	red	red
loss not reported	N/A	pink

Table 1: Color coding by loss.

Finally, if the gateway *mtrace* fails, a *reverse mtrace* is attempted. Sometimes tracing this path yields a result. Using the assumption that the forward path is the same as the reverse path, the route can sometimes be determined. Although this assumption is not always the case, it is better to have a reasonable estimate than having no route data at all. No statistics can be collected for a reverse *mtrace*, because the reverse *mtrace* only displays statistics on data flowing from the receiver to the source. By trying these three different *mtraces*, the successful trace percentage increased substantially. Results showed the average success rate increased from 67% to 80%.

3.2 User Interface

In prototyping, the decision was made to write *MHealth* in Java for the dual reasons of cross-platform operation and ease of GUI prototyping. *MHealth* provides tree visualization functions as well as presentation of other data statistics. These two functions are described next.

3.2.1 Visualizing Multicast Trees

When a user starts *MHealth*, they provide a multicast IP address and a port number, either on the command line or in a startup menu. The user also has the option of enabling logging at this point. The logging function writes to a file all the RTCP packets received and the *mtraces* performed throughout the session. Once started, *MHealth* begins listening for RTCP packets and builds a source and receiver list. This list is displayed on the *MHealth* window as they are identified. The sender(s) are displayed across the top of the screen and the receiver(s) are displayed across the bottom of screen from left to right in the order they are first heard. The domain name of the host is displayed if it fits in the box, otherwise the IP address is displayed. Reception quality information including the packet loss rate and jitter is also obtained from the RTCP packets. As the session sources and receivers are displayed, they are color coded according to their loss percentage (see Table 1). The color code is updated with each new RTCP packet received.

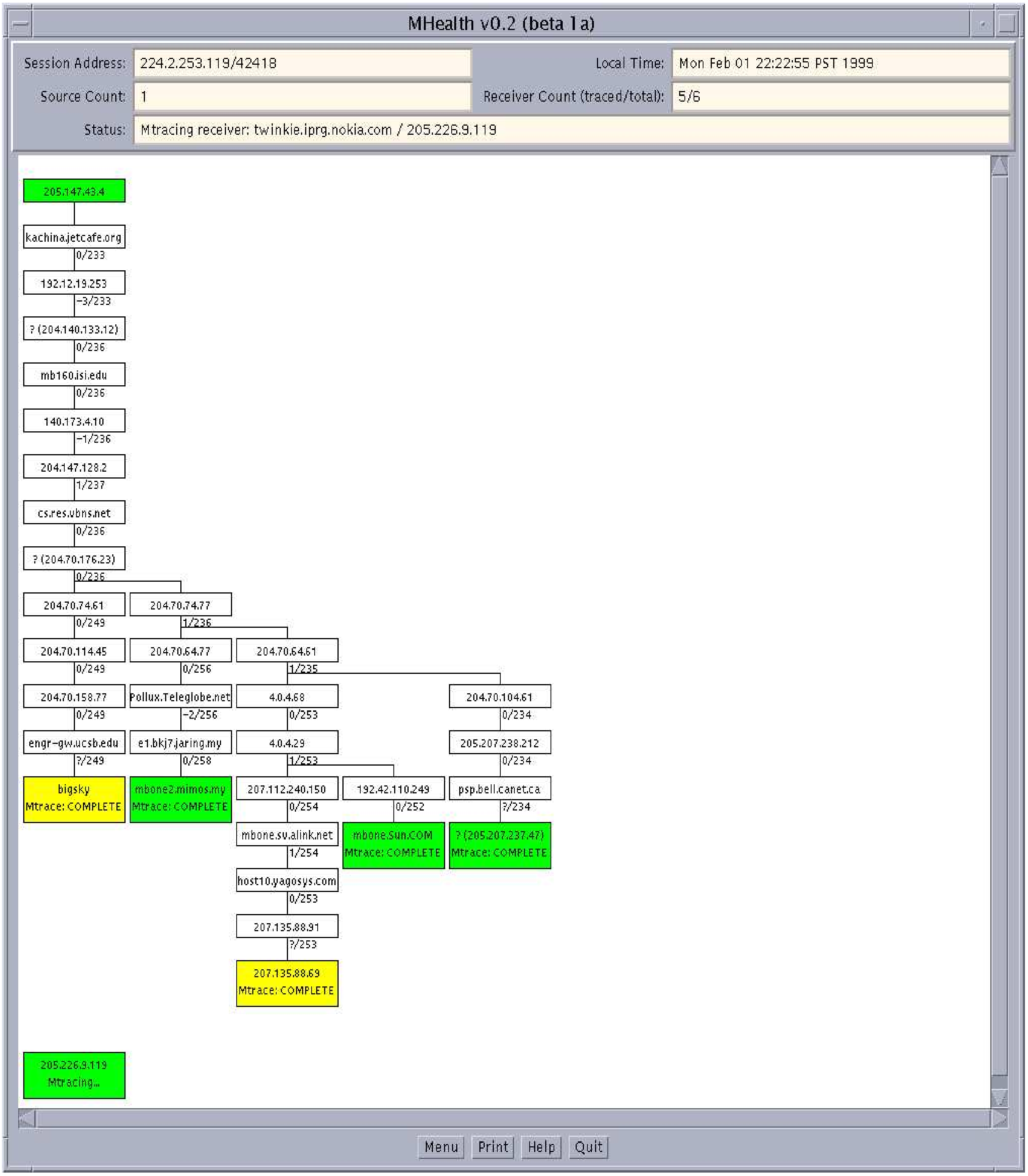
Once *MHealth* has heard from at least one source and receiver, it can start building the multicast tree. *MHealth* begins executing *mtraces* to determine <*source,receiver*> paths. Once a route is determined, the path is drawn graphically on the screen. The *mtrace* packet loss statistics are reported below each hop. These statistics are represented as a fraction of packets lost over total packets expected. Measurements are made for an interval of time measured right before the router was queried, e.g. 5/265 would indicate that 265 packets were expected but only 260 were received. The computed percent loss is used to color code the routers in the same way that senders and receivers are coded. One difference is that routers with no or low loss are colored white instead of green to distinguish them from end hosts.

Occasionally *mtrace* will report a negative number of packets lost, such as -5/265. This -5 indicates that the router actually received 270 packets out of 265 expected. Extra packets are probably due to unnecessary router duplication. Every once in a while, packet duplicates will be significantly larger, possibly suggesting flooding or routing loops are occurring. An important note is that packet duplication for any reason can mask loss. That is, -5/265 could mean that no packets were lost and 5 packets were duplicated, or it could mean that 10 packets were lost and 15 packets were duplicated. There is no way to differentiate these two cases. *MHealth* represents a negative packet loss as no loss.

Figure 1 shows a snapshot of the *MHealth* tool. The tool has displayed a small multicast tree. Once *MHealth* has traced all receivers in a session, it loops back to the beginning of the receiver list and continues repeating *mtraces* to keep the loss data and routes reasonably up-to-date.

Because *mtrace* is a point-to-point path discovery tool, there are some issues in combining its results into a tree. The first issue is how to deal with different reported packet statistics for overlapping links in the tree. In this case, *MHealth* keeps only the most recent statistics. As each new *mtrace* is executed and its results updated in the tree, it will overwrite any existing hop data on shared tree nodes. Since all of these receivers have at least one hop in common, some of the data overwritten will be from other receivers. In Figure 1, for example, the last receiver traced was the one on the far right, 205.207.237.47. Therefore, the packet loss statistics for the shared links in the path to 205.208.237.47 are actually the loss reported from the *mtrace* to 205.208.237.47.

Over time, the membership and topology of the group may change. As RTCP packets are heard from newly joined receivers, they are added to the bottom of the *MHealth* window, and will eventually be traced. If an explicit RTCP *BYE* packet is received, the receiver and the portion of the topology that was unique to it will be immediately removed from the tree. In the case where a *BYE* packet is lost, the receiver will eventually be timed out of the session. After the first few



Session Address: 224.2.253.119/42418

Local Time: Mon Feb 01 22:22:55 PST 1999

Source Count: 1

Receiver Count (traced/total): 5/6

Status: Mtracing receiver: twinkie.iprg.nokia.com / 205.226.9.119

Figure 1: A sample *MHealth* screen shot of a small multicast tree.

minutes without receiving a packet, the receiver's box will turn gray. After a few more minutes, the receiver and their associated topology will be removed from the tree.

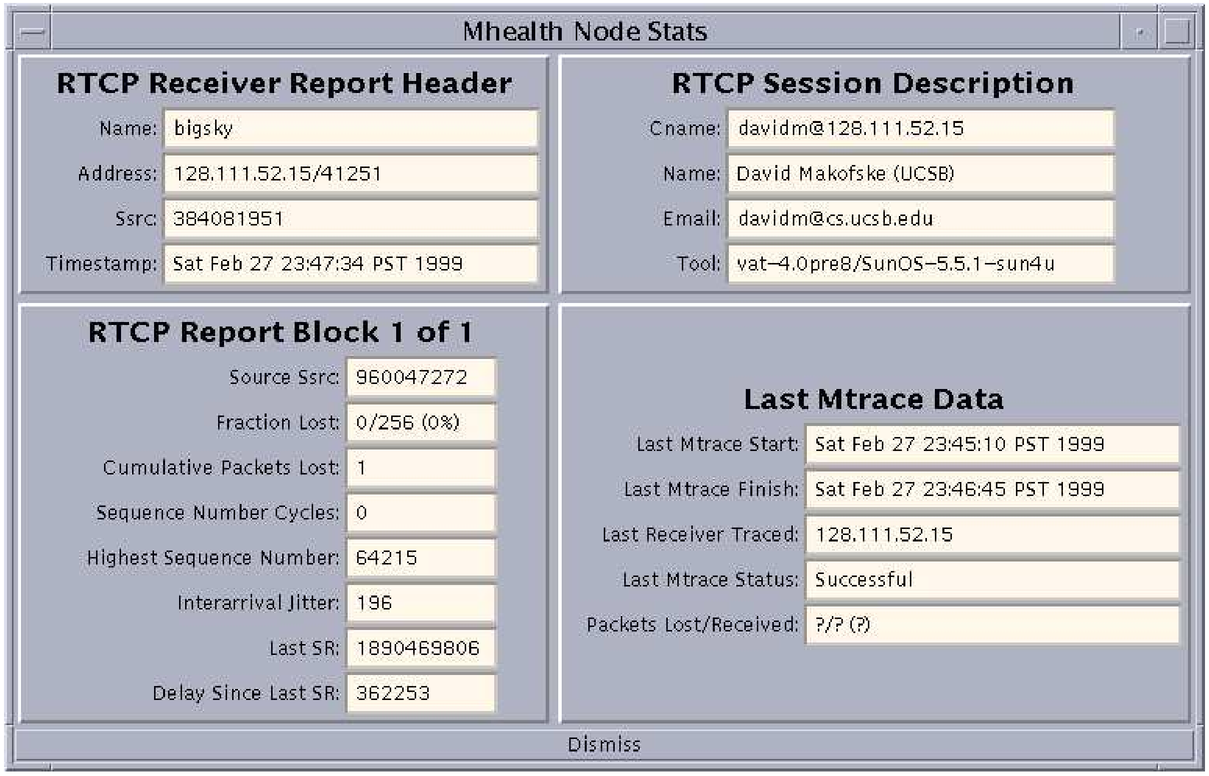
Topological changes may occur as a receiver is traced multiple times. Initially the idea was to keep all topology information in the tree, so that route flapping and changes could be visually identified. However, when route flapping occurred frequently, this made the tree confusing and distracting. The approach was changed to only show the most recent topology traced for each receiver. Route flapping and route changes are relegated to log files for post-session analysis.

3.2.2 Viewing Additional *MHealth* Information

In addition to visualizing a group's tree, *MHealth* is interactive and provides a number of user functions. Each of these functions is described below.

View Stats. Additional information for each node in the tree can by displayed by clicking on the node. A window of collected data for that node is displayed. The actual data displayed will vary depending on the type of node: sender, receiver, or router. Senders and receivers always display the data from the most recently received RTCP packet. This information is broken into three basic display sections, which roughly correspond to the sections of the RTCP packet itself. If *mtrace* data is available (the node has been traced at least once), a fourth section with *mtrace* data is also displayed. Figure 2 shows a sample statistics window. A window has all of the following parts:

- The first section is the report header. This contains the source's host name, IP address and port, and a timestamp of its receipt at the local host.
- The second section is the *source description*. This includes various textual information about the packet sender. The most commonly transferred values are the canonical name, email address, the person's name, and the tool used.
- The third section is the *report block*. There may be a variable number of report blocks from 0 to 32. Each report block corresponds to a single received data stream. If there is a single source, there will only be one report block. These report blocks may appear in any order the transmitting tool chooses. No report blocks will be sent if all the sources in the session have ceased transmitting, or if the receiver has lost all connectivity (either due to a total failure or heavy congestion). The most important data in the report block is the fraction of packets lost. This value is an an eight bit number representing the fraction of RTP data packets received out of 256 since the last report was sent. This fraction is calculated as the number of packets lost divided by the number of packets expected as determined by the RTP sequence numbers.
- The fourth section is the *mtrace block*. If an *mtrace* has been successfully executed, statistics are displayed including the timestamp that the *mtrace* was started and completed, the receiver that was being traced, and whether the trace was successful.



RTCP Receiver Report Header

Name:	bigsky
Address:	128.111.52.15/41251
Ssrc:	384081951
Timestamp:	Sat Feb 27 23:47:34 PST 1999

RTCP Session Description

Cname:	davidm@128.111.52.15
Name:	David Makofske (UCSB)
Email:	davidm@cs.ucsb.edu
Tool:	vat–4.0pre8/SunOS–5.5.1–sun4u

RTCP Report Block 1 of 1

Source Ssrc:	960047272
Fraction Lost:	0/256 (0%)
Cumulative Packets Lost:	1
Sequence Number Cycles:	0
Highest Sequence Number:	64215
Interarrival Jitter:	196
Last SR:	1890469806
Delay Since Last SR:	362253

Last Mtrace Data

Last Mtrace Start:	Sat Feb 27 23:45:10 PST 1999
Last Mtrace Finish:	Sat Feb 27 23:46:45 PST 1999
Last Receiver Traced:	128.111.52.15
Last Mtrace Status:	Successful
Packets Lost/Received:	?/? (?)

Figure 2: Receiver statistics from RTCP and *mtrace* data.

Pruning and Expanding Nodes. Routers within the tree can be hidden or expanded to create a custom tree view. This is useful if a user is only interested in the traffic within a subset of the multicast tree. After a router has been pruned from the window, the router above will display a small green bar along its base, visually indicating that there are routers and receivers below which are not visible. On routers where there are pruned nodes below, their menu option for “Prune” is replaced by “Expand”, allowing those downstream routers and participants to be re-displayed.

Changing Senders. *MHealth* is only capable of representing a multicast tree for a single source at a time. If more than one source is detected in a multicast session (which would be the case in a many-to-many multicast session like a video conference), the senders are placed from left to right across the top of the screen. *MHealth* is not designed to handle multiple sources because each sender will have its own unique multicast tree to the group’s receivers. Since *MHealth* can only display a single multicast tree at a time, it chooses the first source heard as the tree root. For each additional source displayed to the right of the root, their menu will display an option to “Make Root”. When this option is selected, the current root and the selected source will swap positions, and the current tree will be discarded. Every receiver will be returned to their position at the bottom of the window, and *mtraces* will begin anew to place them into the new tree, rooted at the new source.

User *Mtrace* Control. The "Mtrace Next" option can be used on any receiver in the multicast group. This option alters the normal *mtrace* pattern. Once a group of receivers is identified, *MHealth mtraces* them one by one in the order they were detected. When the last group receiver is traced, the process begins again at the beginning of the receiver list. At any time, if the "Mtrace Next" option is selected for a receiver (whether it has been placed in the tree or not), it will become the next node to be traced. The text "Mtrace: SCHEDULED" will be placed inside the node to indicate the action has been handled. When the currently executing receiver trace has completed, the scheduled node will be traced. The trace order will then continue back to the node that would have been scheduled prior to the user intervention. This order is used to preserve uniform handling of new receivers. This is important both for presenting an accurate representation of the tree but also for accurate collection of statistics. Only one node may be scheduled in this manner at a time. If another *mtrace* is scheduled before the prior scheduled one has started, the prior scheduled *mtrace* will be cancelled, and the text "Mtrace: CANCELLED" will be displayed in the receiver node.

3.3 Sample Analysis Using *MHealth*

MHealth provides a fairly quick view of small and medium-sized trees. As an example of analysis that can be performed with archived *MHealth* data, we present results collected during one week in December 1998. During that week, both the 43rd Internet Engineering Task Force (IETF) meeting and NASA TV were being broadcast. Our goal in using *MHealth* was to observe its ability to quickly, efficiently, and effectively trace a medium-sized group. We also used the results to observe the stability of the multicast forwarding tree used by these groups.

Over the two week period that we used *MHealth*, receivers were traced in an average of 41 seconds. Also, 19% of the total trace attempts failed completely. In attempting to understand why these traces failed, we believe that there were at least two primary reasons. First, *mtraces* failed due to older routers which do not implement the proper functionality. Second, congested routers place a low priority on responding to *mtrace* queries and may not respond within the timeout period. Of course, the other main reason why *mtraces* could have failed is in routing or forwarding problems. In these cases receivers likely were not seeing the group's traffic.

As an example of the kinds of post-event analysis that can be performed, we used the archived *MHealth* data to study the number of distinct routes for each group receiver. Figure 3 shows the results from this analysis; two data sets are included. We sub-divide each set of results into two types. Because of the way *MHealth* works and the variability in member lifetimes, some receivers are only traced once while others are traced numerous times. This typically happens when a receiver is only part of the group for a short period of time. In such a case, a receiver will only have one possible route. Therefore, for each data set, there are two graphs. The left graph includes all receivers, and the right graph eliminates these "singly-traced" receivers, called long-lived hosts, from consideration. Results show that a large number of receivers for both the IETF and the NASA data sets had a significant number of alternate routes; some even as high as 14 different routes over the course of a week. In the right graph approximately 70% of IETF group members and 45% of NASA group members had at least one route. One possible reason for the unusually high route variability for the IETF session was because network engineers were testing new inter-domain protocols and made changes to the local topology several times. Additional analysis is beyond the scope of this paper and is left to future work.

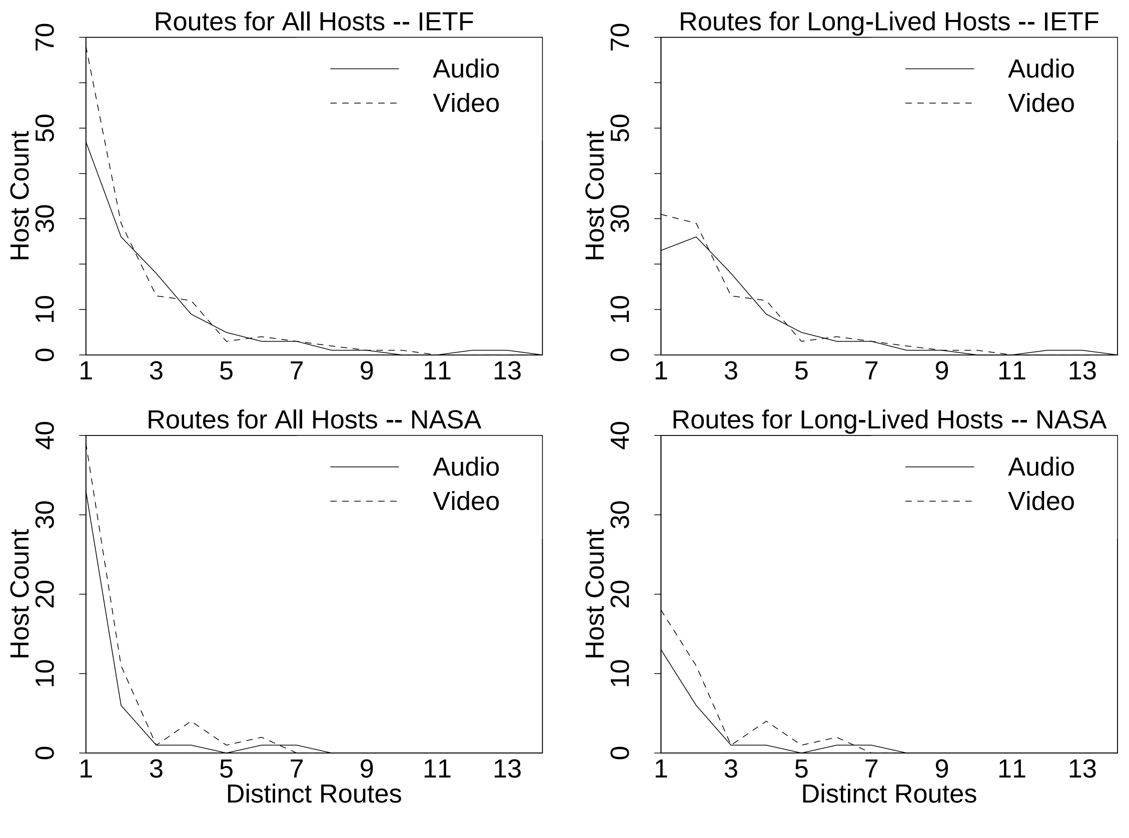


Figure 3: Number of distinct routes recorded for each group member. Breakdown by data type (IETF/NASA) and number of traces per host (All/Long-Lived).

4 Evaluation of the *MHealth* Tool

MHealth provides important functionality for monitoring multicast groups, but the question is whether *MHealth* works well enough to be an acceptable solution for network administrators. In answering this question, the three most important issues were determined to be *MHealth's* scalability, its granularity of data, and its overhead.

4.1 Scalability and Granularity of Data Collection

MHealth has a number of critical scalability issues. The most important of these are associated with the collection tools themselves. Both RTCP and *mtrace* fall short of providing the actual data needed.

RTCP. Fewer RTCP packets per participant are sent as group size increases. This prevents overwhelming a large multicast session with control data. Because the packet loss information included in an RTCP packet covers all data received since the last RTCP packet sent, no period of time will be missed. However, the granularity of the data collected will be reduced. As a result, although *MHealth* will receive RTCP packets at roughly the same rate regardless of group size, the frequency of the updates for each individual receiver decreases.

Another major RTCP concern is whether the packets will actually be received at all. There are a number of potential reasons why this issue needs to be considered. First, RTCP packets are over UDP, and therefore unreliable. These packets can potentially be lost and will likely be lost in highly congested networks. Second, RTP tools operating behind a firewall may not be able to get any of their control packets to other group members. Finally, there are some streaming media tools that do not implement RTP. In fact, some tools which do implement RTP do not implement RTCP or fail to implement it properly. The net result is that sometimes the worst performing group members have the hardest time telling the group of their performance problems. While RTCP meets its design goals, it falls short as an accurate accounting tool.

***Mtrace*.** *Mtrace* has similar scalability problems. Tracing a route takes a certain length of time to complete (averaging approximately 41 seconds during one two-week test). Since the receiver list is sequentially traversed, the larger the list of receivers, the more time elapses between *mtrace* attempts. And again, congestion problems in the network will cause traces to fail. No information will be provided for group members whose network problems need to be addressed.

It is not necessarily clear what can be done to address these scalability issues. RTCP or any application-level feedback needs to reduce its granularity as the session membership grows. Unless some management station is configured and capable of receiving hundreds or thousands of messages per second, multicast feedback will not scale. *Mtrace* data collection can be enhanced by executing multiple *mtraces* simultaneously, but then there is a risk of adding too much overhead to the routers. There may be some methods to reduce the *mtrace* overhead by doing more intelligent tracing. This topic is addressed more in the next section and in future work.

4.2 The Overhead of *mtraces*

The overhead of tracing a multicast session's topology in a repeated and automated way is a potential concern. Every successful *mtrace* requires a response from every router along the path from the receiver to the source. There are two concerns about this type of overhead. First, there are concerns for a single *MHealth* monitor running for a group. Second, the concern is even greater for a single group that has multiple or many *MHealth* monitors running.

Single ***MHealth*****.** If the session participants are reasonably distributed (meaning that a large number of them do not share the same last hop router), successive *mtraces* initiated by *MHealth* will be distributed throughout the topology. This means that the routers closer to the leafs in a reasonably distributed group will not be required to respond to repeated *mtrace* requests. The larger concern, however, is for the routers closer to the source, especially the first hop router from the source, which must respond to every *mtrace*. For these routers, the interval between when they must respond to *mtrace* requests is directly proportional to the time it takes to complete an *mtrace*. A valid concern is that the frequency with which these routers must respond to *mtrace* packets in addition to their normal unicast and multicast routing duties may actually cause more congestion. Two potential solutions are:

1. By design, *mtrace* IGMP packets may be ignored when the router is under heavy load[5]. This should prevent *mtrace* from overloading a router in most cases. However, it would be poor design for a network monitoring tool to rely on router robustness to prevent congestion. One possibility is to apply an exponential backoff of the frequency of *mtraces* when router congestion is detected, at the expense of trace information.

2. Another approach is to only trace from the receiver to the boundary of the known tree instead of all the way to the source. A full *mtrace* would be run back to the source periodically, but less frequently. This would allow the tree topology and statistics to still be updated, but would reduce the frequency of trace updates on the portions of the tree closer to the source. This approach is discussed further in the future work section.

Multiple *MHealths*. Another perspective that must be considered is multiple copies of *MHealth* running simultaneously for the same session. Other than requiring a *setuid* of root for running the *mtrace* tool on Unix systems, there are no limitations to running *MHealth* on any session. Since we have released *MHealth* as a freeware tool, the possibility exists that many copies of *MHealth* could be run simultaneously on the same session from different locations without knowledge of each other. In an extreme case of a small number of participants and/or a large number of *MHealth* users, every router in the tree could potentially need to respond to *mtrace* requests on an almost constant basis. The approaches for reducing *mtrace* congestion discussed from a stand-alone perspective above could be applied in this situation as well. Two additional solutions also have been considered. The first is to integrate *MHealth* into a web browser and use a single, central collection point but then distribute results to interested sites. The second is to make *MHealth* passive and collect statistics by listening to others conducting traces. Both of these schemes are described in the next section.

5 Future Work

The items listed in this section are proposed from the perspective of offering criticisms about *MHealth* and then offering potential solutions. While not perfect, *MHealth* offers a valuable integration of existing collection tools and visualization techniques. In listing the most common problems, the majority have to do with the overhead of *mtraces* and the problem of having multiple instances of *MHealth* running for the same group at the same time.

5.1 Web Integration

One of the best ways to reduce the overhead of doing an *mtrace* in a multicast session is to limit the number of people who are doing *mtraces*. One way to do this is to only have a single user actually running the *MHealth* program, and have that user create a display of the *MHealth* data on a web site. Other interested parties would simply access the data that way.

An implementation of this was attempted using a Java applet and tested during the 42nd IETF in August 1998. When the Java applet was loaded into the web browser, it contacted the instance of *MHealth* via a unicast TCP socket. Due to Java applet security restrictions, the *MHealth* controller was required to run on the same machine as the web server. The *MHealth* application would then transfer a serialized copy of the entire tree structure and all of its associated data to the Java applet. The applet would display the graph in an identical fashion to the application itself. The applet maintained full interactivity, allowing clicking on nodes for RTCP and *mtrace* statistics, and the pruning and expanding of nodes. A "refresh" button in the applet recontacted the *MHealth* application to receive a new snapshot of the data.

Although the applet worked well in a controlled environment with small trees and few requests, deployment testing proved this approach to be unscalable. As the quantity of tree data and the number of requests increased, both the *MHealth* application and the applets either crashed or displayed erratic results. In addition, the variety of operating systems, browser versions, and Java versions created a number of bugs that did not occur when the Java virtual machine versions and implementations were carefully controlled.

As an alternative to an active Java applet, another approach that has been considered is the generation of a GIF or JPEG image of the *MHealth* tree. This image would be placed in a web page and updated frequently. The drawback is that interactivity and additional information requests would not be supported. This may or may not be an acceptable alternative for users, but it does offer a nice tradeoff in that simplified functionality reduces complexity.

5.2 Passive Mode *MHealth*

One interesting solution for the problem of of multiple *MHealth* tools running for the same session is to utilize *passive mtraces*. The results of most *mtraces* are multicast onto a well-known multicast address. A passive *mtrace* would not send an *mtrace Query*, but promiscuously listens for an *mtrace* which matches the needed query. An implementation of *MHealth* which uses passive *mtraces* could have an active and a passive mode. Passive mode would be the default, and an *MHealth* process would listen for any *mtrace* that matched one of the *mtraces* it would need to know about. As long as needed *mtraces* were being received, *MHealth* would stay in passive mode. If no relevant *mtraces* were heard after some (possibly random) timeout period, *MHealth* would enter active mode and begin sending *mtrace* queries. In theory, this would create a single "leader" *MHealth* process that would send queries, and the other *MHealth* processes would remain passive.

5.3 Total *Mtrace* Control and Partial Tree Tracing

Another *MHealth* optimization would be to integrate the *mtrace* functionality entirely in the Java application. This would be an effective alternative to the current method of parsing the output from the existing *mtrace* tool. Reliability would be greatly enhanced and *MHealth* would have grater control over traces. Additional features such as a separate window for watching trace activity could also be added.

Tighter trace control would also allow for partial tree tracing. In order to reduce overhead, traces could be initiated from the receiver into a known portion of the tree, rather than completely to the source. This would create a less up-to-date tree at the nodes closer to source, but would reduce IGMP query overhead at the source. Periodic traces could be done all the way to the source in order to update the nodes closer to the root.

5.4 Customizable Trace Behavior

MHealth could benefit from a more flexible trace scheduling algorithm. Currently, users are limited to the selection of the next node to be traced. Additional functionality might allow the user to choose a scheduling heuristic. For example, a user might want *mtrace* to focus on receivers which have not yet been added to the tree, or the user could focus on giving more attention to dealing with problem cases. Our log statistics showed that, on average, only 60% of the total number of unique IP addresses discovered with RTCP packets stayed in the session long enough to attempt at least one trace. Often what happened was a new user joined the group but left before the user could be traced and added into the tree. This suggests that tracing new members first would improve the breadth of data collected.

5.5 Session Playback

One limitation to *MHealth* is that the display only provides a real-time snapshot of activity. While this is useful for real-time monitoring, it limits the flexibility of post processing and visualization. An enhancement would be the ability to replay a logged *MHealth* session. Since all of the data used to draw the tree can be logged (the RTCP packets and the resulting *mtraces*), it would be possible to watch how the multicast tree changes over time. This would be done after the session is over, and would utilize "time-lapse" playback. This would be useful for loosely observing group and tree behavior over time, and also for debugging problems after a session had ended.

6 Conclusion

This paper first points to the advantages of multicast as a paradigm for improving bandwidth usage, especially for broadcast-style, streaming audio and video traffic. Multicast is growing in popularity, and Internet deployment efforts have been growing as well. There is a clear need for tools to monitor multicast traffic, diagnosis faults, and analyze traffic flow. This need is often cited as one of the most significant barriers to widespread multicast adoption.

MHealth is a graphical, near real-time multicast monitoring tool developed to address multicast traffic management needs. *MHealth* allows a user to view and collect data about the health and topology of a multicast tree. An important aspect of *MHealth* is its ability to integrate data from multiple sources and provide a more comprehensive view of multicast data flows. Despite concerns about scalability and granularity of data collection, *MHealth* has been shown to be a useful tool for collecting, processing, and archiving topology data.

References

[1] D. LeGall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, pp. 47–58, April 1991.

[2] K. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment," *IEEE Network*, January/February 2000.

[3] C. Diot, B. Lyles, B. Levine, and H. Kassem, "Requirements for the definition of new IP-multicast services," *IEEE Network*, January/February 2000.

[4] K. Almeroth, "Managing IP multicast traffic: A first look at the issues, tools, and challenges." IP Multicast Initiative White Paper, August 1999.

[5] W. Fenner and S. Casner, "A 'traceroute' facility for IP multicast." Internet Engineering Task Force (IETF), draft-ietf-idmr-traceroute-ipm-*.txt, August 1998.

[6] H. Schulzrinne, S. Casner, R. Frederick, and J. V., "RTP: A transport protocol for real-time applications." Internet Engineering Task Force (IETF), RFC 1889, January 1996.

[7] A. Swan and D. Bacher, *rtpmon 1.0a7*. University of California at Berkeley, January 1997. ftp://mm-ftp.cs.berkeley.edu/pub/rtpmon/.

[8] K. Almeroth, *Multicast Group Membership Collection Tool (mlisten)*. Georgia Institute of Technology, September 1996. http://www.cc.gatech.edu/computing/Telecomm/mbone/.

[9] J. Robinson and J. Stewart, *MultiMON 2.0 – Multicast Network Monitor*, August 1998. http://www.merci.crc.ca/mbone/MultiMON/.

[10] A. Rubens, C. Ravishankar, D. Thaler, A. Adams, B. Norton, and J. DiGiuseppe, "Merit SNMP-based MBone management project." http://www.merit.edu/~mbone/.

[11] M. Handley, "An examination of MBone performance," Tech. Rep. ISI/RR-97-450, Information Sciences Institute (ISI), University of Southern California (USC), January 1997.

[12] M. Yajnik, J. Kurose, and D. Towsley, "Packet loss correlation in the MBone multicast network," in *IEEE Global Internet Conference*, (London, ENGLAND), November 1996.

[13] K. Almeroth and M. Ammar, "Multicast group behavior in the Internet's multicast backbone (MBone)," *IEEE Communications*, vol. 35, pp. 224–229, June 1997.

[14] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," *ACM Sigcomm*, pp. 200–208, September 1990.

[15] J. Nagle, "Congestion control in IP/TCP internetworks." Internet Engineering Task Force (IETF), RFC 896, January 1984.

[16] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol (DVMRP)." Internet Engineering Task Force (IETF), RFC 1075, November 1988.

[17] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, G. Liu, and L. Wei, "PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, pp. 153–162, Apr 1996.

[18] J. Moy, "Multicast extensions to OSPF." Internet Engineering Task Force (IETF), RFC 1584, March 1994.

[19] W. Fenner, "Internet group management protocol, version 2." Internet Engineering Task Force (IETF), RFC 2236, November 1997.