MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool*

David B. Makofske
Starburst Software †
150 Baker Avenue Extension
Concord, MA 01743-2117
david_makofske@starburstsoftware.com

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

Abstract

The exponential growth of the Internet combined with the increasing popularity of streaming audio and video content are pushing Internet bandwidth constraints to their limits. Methods of managing and more efficiently utilizing the existing bandwidth are becoming increasingly vital. Multicasting Internet content, especially streaming audio and video, can provide enormous bandwidth savings. A decade of efforts at deploying multicast combined with the rising need for better traffic management for bandwidth-hungry audio and video applications has led to significant momentum for multicast implementation. One of the remaining barriers to widespread adoption of multicast in the Internet 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 Real-Time 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.

1. Introduction

Recent exponential growth in the Internet has brought the total number of connected people up to 70 million at the end 1998. 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 net result is that an increase in both the number of users and the amount of data transfered has produced considerable strain on the existing Internet. This has led to a number of highly publicized recent network and server failures when major events such as political scandals, international crises, or entertainment events attracted users to request streaming audio/video content en mass. The growth trends suggest these events, and the congestion they cause, will continue to grow.

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. All of this can be coupled with the rapidly increasing bandwidth capacities in the "last-mile", one of the few remaining bottlenecks helping to restrict user data flow into 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. Bandwidth management takes 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 model for streaming audio and video. This is especially true when it is used in broadcast-style applications, i.e. where 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 to a subset 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 that multicast address. 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

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

†This work was performed as part of David Makofske's Masters Thesis at the University of California, Santa Barbara.

the same network link more than once. With high quality MPEG at several Mbps per stream, this would greatly reduce the load on the server and the network, especially as the number of receivers increased into the hundreds, thousands, or more. Depending on the location and number of receivers, the bandwidth saved with multicast over unicast can be many orders of magnitude.

Despite the conceptual simplicity of multicast and its obvious benefits, its implementation is difficult. After almost a decade of research and implementation efforts, multicast has been deployed as the research-oriented Multicast Backbone (MBone)[1, 2]. It is only beginning to be deployed by Internet Service Providers (ISPs) and used by service-oriented companies. 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) have been working to educate consumers about the benefits of multicast, and major ISPs such as UUNet and Sprint have been implementing and promoting multicast as a network service. Broadcast.com, a leading provider of streaming media content on the Web, has been actively encouraging the use of multicast, and commercial tools such as Real Network's RealVideo and Microsoft's NetShow are now multicast enabled. Most router manufacturers now include multicast capabilities as a standard feature on their routers.

Many ISPs are still reluctant to enable multicast on their networks, however, because of its complexity and fundamental differences from unicast. There are few tools available for monitoring and debugging multicast networks, and the tools that do exist are understood by only a few experts. In addition, many of the current tools mimic unicast tools, and provide information on a single end-to-end path from a source to a receiver instead of the entire multicast tree. Others concentrate on individual participant information only and ignore the routes entirely. 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 architecture. While some tools for multicast monitoring exist, there are no tools which work in today's Internet and provide both tree visualization and distribution quality 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 an MBone 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 network tree distribution and delivery quality. MHealth also provides data logging functionality for the purpose of isolating and analyzing network faults.

The remainder of this paper is organized as follows. Section 2 discusses related work in the field of multicast network monitoring and MBone data analysis. Section 3 introduces the MHealth 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. 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 begun to lose some of their utility. Today's system administrators need tools more suited for identifying whether multicast traffic and trees are “healthy”, i.e. whether a group's traffic is of acceptable quality and whether traffic is reaching all of the group's members and no one else.

One of the most important tools in use today is *mtrace*[3], a multicast version of traceroute. *Mtrace* works by starting at the receiver and traces the *reverse* path back to the source. The reverse path is used since this is how group members are added to the group; receivers send join requests toward the source using the path the receiver would use to reach the source. Mtrace is one of the primary data sources used by MHealth to visualize a multicast tree, and will be discussed in detail later in the paper.

Several tools provide statistics based on Real-Time (Control) Protocol (RT(C)P)[4] packets. The MBone *vat* tool is one of the primary MBone audio tools and it uses RTCP packets to create a list of group members. *RTPmon*[5] collects and displays this information along with real-time statistics about packet loss and jitter. *Mlisten*[6] uses RTCP packets to collect and archive group statistics for all MBone groups advertised through *sdr*. MultiMON[7] monitors the multicast traffic on a local network segment and provides graphs on traffic type 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*. *Mstat*[8] allows an SNMP-enabled router to be queried for information, including routing tables and packet statistics. *Mrtree*[9] uses cascaded SNMP router queries to provide a text-based representation of a particular multicast group's topology. *Mview*[10] is a tool for visualizing MBone topology as well as monitoring and collecting *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 because a system administrator will have the proper passwords. Unfortunately, the utility of these tools is somewhat limited in the inter-domain.

Several papers have been published on the analysis and behavior of multicast groups in the MBone. In work by Handley, tools were written to log RTP and RTCP packets and collect *mtraces* for an MBone session[11]. These were later used to manually create a picture of the multicast tree and various statistics about the links in the tree. There were no results presented in the paper about how these techniques could be extended to handle real-time operation.

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.

3. The MHealth Tool

In this section we present the MHealth tool. First the design of MHealth and how it integrates the available data sources is discussed. The user interface of MHealth is then presented, and the section concludes with observational analysis of MHealth performance.

3.1. Design

The design of MHealth 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, TCP is not a suitable protocol. 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 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. 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 addition, real-time data has very specific requirements in terms of inter-arrival timing beyond what even TCP can provide. And while data should be delivered as accurately as possible, the real-time nature of streaming media prohibits retransmission of lost packets.

As a solution to these problems, real-time streaming data (both unicast and multicast) typically uses an application level protocol, sometimes called an Application Layer Framing (ALF)[14] protocol. An ALF 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 protocols add important functionality such as reordering and packet timing on top of UDP's sequence numbers and port multiplexing services.

One of the most common ALF protocols used for streaming real-time audio and video is the Real Time Protocol (RTP)[4]. RTP 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. RTP includes a control protocol called the Real Time Control Protocol (RTCP), which allows for participants in a multicast group to report their membership and the quality of their reception.

Multicast monitors cannot rely on end-to-end state for network monitoring the way TCP-based tools can. There is no transport level connection setup that can be used to establish the participants in a multicast group. The state in multicast networks is distributed among the group participants and network elements, so alternative mechanisms must be utilized to do monitoring. There are two basic entities to gain multicast group information from: the routers, and the participants themselves. Participants can only provide data at the application level, which is precisely the type of feedback that RTCP was designed to provide. Routers can be queried directly by using SNMP or by dumping and examining their state tables. Although, for security reasons, this is unlikely to be possible on a end-to-end basis in the Internet. Routers can also be indirectly queried through a traceroute-based facility like *mtrace*. Many existing tools have relied exclusively on one approach or the other, either collecting only application level feedback or collecting only router-based information. Tools that do use both methods typically display the results separately and do not attempt to integrate them.

MHealth integrates both application layer data and 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 from each receiver to the source. As the point-to-point topology is established for each receiver, this information is combined into a tree data structure and graphically displayed. The details of these data sources are now described in detail.

3.1.1. Real Time Control Protocol

The Real Time Control Protocol (RTCP) specification is defined as part of the Real Time Protocol (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 participants to all other participants in the session. 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. First, it 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. Second, 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). Third, 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. The fourth function of RTCP is for distribution of group membership information. This is meant to be used in a "loosely controlled" manner, however, and 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 one *receiver report* per RTP data source, up to a maximum of 32. These reports contain reception statistics about each data source. The *source description* section contains one or more text elements, including the canonical name, describing 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 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 was not controlled, it would grow linearly with the number of group members. For large groups, this could swamp the group with control information in a manner similar to a group-wide ACK implosion. To avoid this problem, each group member uses an algorithm to determine an RTCP send interval. This interval decreases as the group membership increases. In small sessions RTCP packets are usually sent in 5 second intervals, but the algorithm dictates that the RTCP packet send rate should not exceed 5% of RTP data bandwidth. Of interest to the MHealth tool is the granularity at which RTCP packets are sent. Large groups do not send RTCP reports frequently and so it is difficult to do quality assessment on a fine granularity. Dealing with large groups will be 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, UDP packet loss, or tools that do not conform to the RTP specification. However, RTCP packets are the only way to determine group membership of an MBone session without dumping router statistics. Currently, RTCP is the best mechanism available for a tool like MHealth to ascertain group membership information.

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[3]. One of the disadvantages of *mtrace* is that it only gives you the route from one participant to the source. So MHealth must perform numerous *mtraces*, one for each participant in a session. The tree topology is then constructed by combining the common nodes. *Mtrace* also provides additional information such as total multicast packets per hop, group-specific hop counts, and packet loss per hop. This information is collected and displayed in MHealth.

Although the output from *mtrace* is clearly analogous to 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.

This approach cannot be used in multicast because ICMP TTL expiration messages are explicitly suppressed. This is necessary because of the heavy reliance of IP multicast

on TTL scoping. If ICMP messages were not suppressed, they would likely flood the source of a large group. This necessitated 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 an important tool and crucial diagnostic information.

Because of the way multicast trees are constructed, the tracing mechanisms for multicast are substantially different than the techniques used for tracing a unicast path. Each router in a multicast tree does not know who its receivers are, 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, and by the Internet Group Management Protocol (IGMP)[16, 17, 18] for the last hop (leaf) router. As a result, it is not possible to use the unicast traceroute method of tracing from a source to a particular receiver. At each router only the outgoing interfaces that the traffic should flow on is known. Which of those outgoing paths a given receiver is on cannot be determined. The solution is to start at the receiver's location, and travel backwards towards the source. Since each router knows the incoming interface (from the source) of a group's data, the path can be determined hop-by-hop from the receiver to the source. This is called a *reverse path lookup*.

The reverse path lookup requires that routers be able to process a special *mtrace* IGMP *Query* packet. This packet is multicast on the ALL-ROUTERS multicast address (224.0.0.2). The last hop router of the receiver identifies that it is the last hop router for a particular *Query* packet received and begins an *mtrace*. The last-hop router appends its data to the 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, 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. Data for each router is appended to the *Request* packet.

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 because it would not require an installed *mtrace* tool.

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*. The net result is an increase in 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 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. However, this requires the *mtrace* initiator to know the last hop router. This is determined 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.

Finally, if the gateway *mtrace* fails, a *reverse mtrace* is attempted. The idea is that the tree from the source to the receiver is the same as from the receiver to the source. Although this is not always the case, it is better 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

Loss	Color
$loss < 2\%$	green
$2\% < loss < 10\%$	yellow
$loss \geq 10\%$	red
loss not reported	pink

Table 1. Color coding by loss.

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. Despite concerns that Java would be too slow for intensive real-time processing, this did not turn out to be the case[1].

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. MHealth then begins listening for RTCP packets and builds a source and receiver list. This list of sources and receivers is displayed on the MHealth window as they are identified. The sender(s) are displayed across the top of the screen and the receivers 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 within the box, otherwise the IP address is displayed. Data quality information in the form of 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 a source and receiver list, it can start building the multicast tree. Immediately after at least one source and receiver are identified, 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 the senders and receivers are coded. One difference is that routers with no or low loss are colored white instead of green to more clearly 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 always interprets 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 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. If the *BYE* packet is lost, the receiver will eventually be timed out of the session. After the first few minutes without receiving a packet, the receiver's box will turn gray as a warning that it is in danger of timing out of the session. 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, and route flapping

[1] All tests and data collection were run on a Sun Sparc Ultra 1.

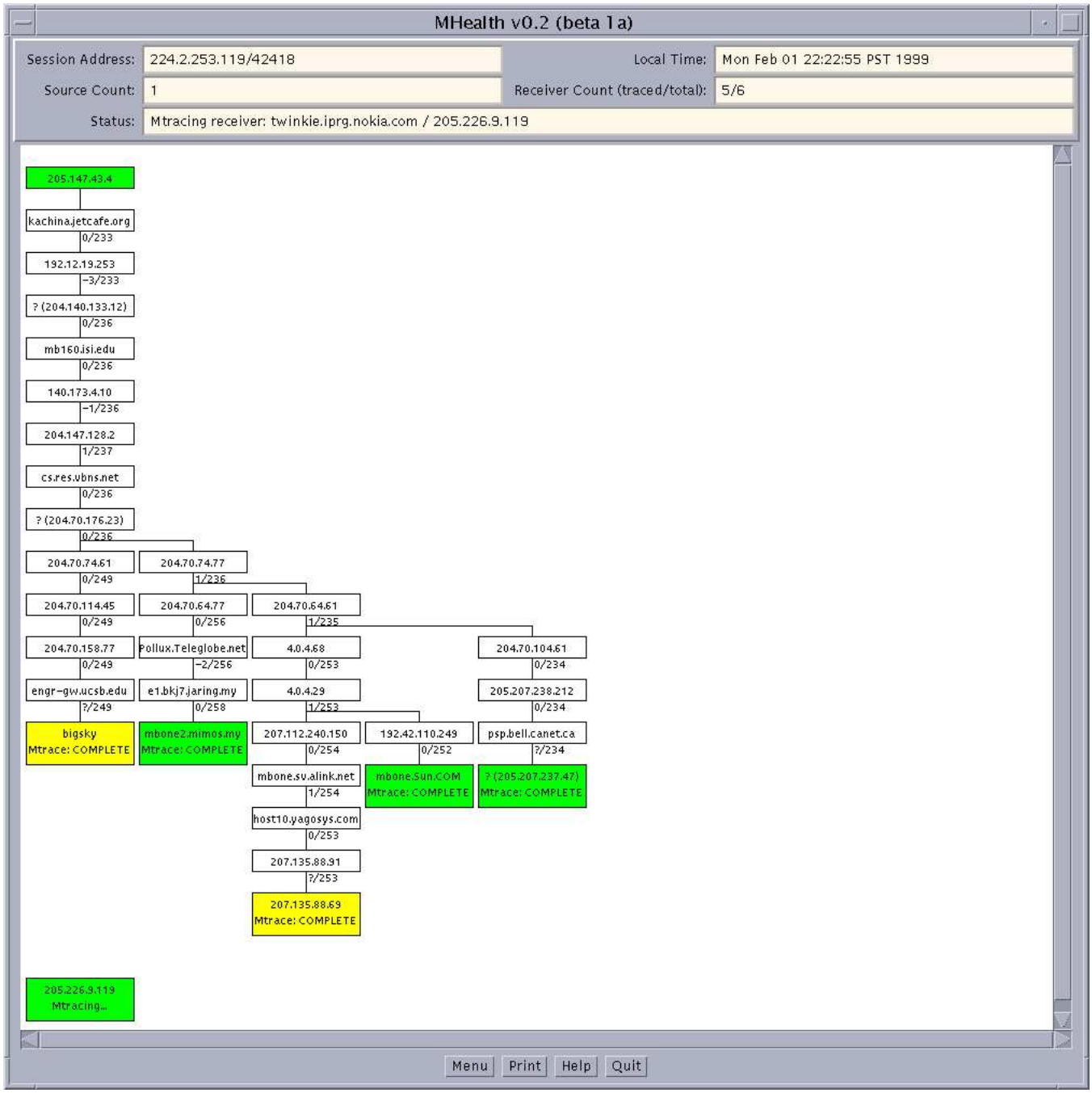

Figure 1. A sample MHealth screen shot of a small multicast tree.

and route changes are relegated to post-session log analysis.

3.3. Interacting with MHealth

Once started with a valid, active multicast group, MHealth begins to draw a multicast tree automatically. No additional user interaction is required. However, MHealth is interactive, and any node can be clicked on at any time. This provides more information about that node. The options and information displayed vary according to the type of node. All nodes have the option ”View Stats”. Receivers have the additional option of ”Mtrace Next”, which allows a user to alter the *mtrace* order. Routers have the options of ”Prune” and ”Expand” to create a custom view of the multicast tree. If there is more than one source detected in a session, users have the option to invoke ”Make Root” on a different source, which will rebuild the tree with the selected source as the new root. Each of these options is now explored in more detail.

3.3.1. View Stats

”View Stats” can be invoked on any type of node to display a window of collected data for that node. The actual data displayed will vary depending on the type of node (sender, receiver, or router) and in some cases on the current state of the node (placed in the tree or not traced). 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 through at least once), a fourth section with *mtrace* data is also displayed. Figure 2 shows a sample statistics window.

- The first section of the packet 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 in the RTCP report 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 in the RTCP report 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. A report block will not be sent at all 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) and believes the sources have stopped transmitting.

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.

If an *mtrace* has been successfully executed for the node that statistics are being viewed on, these statistics are also displayed. Statistics include the timestamp that the *mtrace* was started and completed, the receiver that was being traced, and whether the trace was successful.

3.3.2. Pruning and Expanding Nodes

Routers within the tree can be pruned or expanded to create a custom tree view. This is useful if a user is only interested in the traffic within a small subset of the multicast tree, possibly for debugging purposes. 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.

3.3.3. 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 is obviously the case in many-to-many multicast sessions such as video conferencing), 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.

3.3.4. 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 first 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

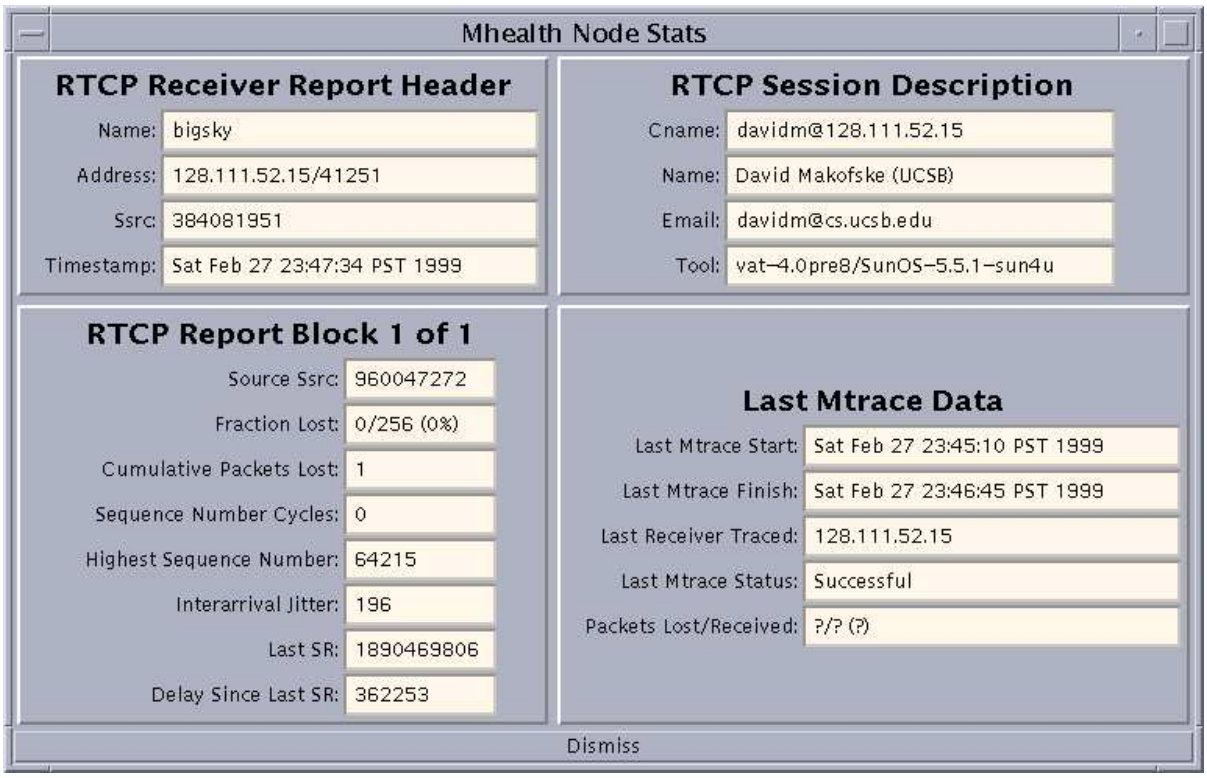

Figure 2. Receiver statistics from RTCP and *mtrace* data.

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.

3.4. MHealth Behavior and Observational Analysis

MHealth provides a fairly quick view of small and medium-sized trees. Over the two week period that we logged data with MHealth, receivers were traced in an average of 41 seconds, and 19% of the total trace attempts failed completely. These failed attempts may be due to older routers which do not implement *mtrace* functionality or congested routers that did not respond within the timeout period.

Viewing the MHealth tree can give a good intuitive feel for the quality of an MBone session, but due to the incremental nature of our data gathering process, the tree never represents an exact snapshot in time. Visually, the RTCP data updates at a much faster rate than the *mtraces*. As a result, the loss colorations along a path in the tree seldom correspond correctly to the loss shown at a given receiver. In addition, the router loss statistics are only consistent along the path of the most recent successful *mtrace*. Because the routers common to the most receivers are all located in the upper portions of the tree, these routers have their *mtrace* data updated more frequently. The further down the branches of the tree, and the further away from the source, the older the loss statistics will be. As a result, the granularity of the router data is finer closer to the source and coarser closer to the leaves.

As long as the user is aware that variations exist in the frequency of the updates from the two data sources, MHealth is fairly intuitive. If the graphical representation of the topology, along with near-real-time statistics offers insight to network managers, the tool will be of value. The combination of RTCP loss data with the tree topology is especially useful in a number of situations. This provides a user with the ability to scan the statistics for all group members and identify any topological influences. A quick inspection for a group of receivers with high loss will tell a user whether all of the loss is caused by the same link. This type of information is difficult to ascertain from any single tool in use today. However, an attempt to interpret

an MHealth tree as an instantaneous representation of the router and flow status throughout the multicast tree would be useless and confusing. While fully real-time updates would be preferred, under the limitations of multicast networks the data granularity achieved may be the best possible without causing substantial network overhead.

4. Evaluation and Issues

The largest issues with MHealth are its scalability and the granularity of its data collection and overhead. Each of these issues is now addressed.

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 packets are sent at a reduced rate as the number of multicast participants increase. 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 decrease as the group size increases.

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.

Mtrace has similar scalability problems. Tracing a route takes a certain length of time to complete (averaging approximately 41 seconds for the data we have logged). 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 and 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 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. Despite these issues, in a connectionless one-to-many network the methods used to collect data are some of the best currently available.

4.2. The Overhead of IGMP Queries

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.

First we look at the concerns from the point of view of a single MHealth process. If the session participants are reasonably distributed (meaning that a large number of them do not share the same last hop router), the successive receiver *mtraces* initiated by MHealth will be distributed throughout the topology. This means that the routers closer to the leaves 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 IGMP packets in addition to their normal unicast and multicast routing duties may actually cause more congestion.

The first thing to consider in this case is that by design, *mtrace* IGMP packets may be ignored when the router is under heavy load[3]. 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.

Another promising approach is to only trace from the receiver into 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.

Another perspective that must be considered is the use of multiple copies of MHealth simultaneously on the same MBone session. Other than requiring a *setuid* of root for running the *mtrace* tool on Unix systems, there are no special requirements for running MHealth on any session. Since it is the intention to release 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. 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

5.1. Web Integration

One of the best ways to reduce the overhead of doing an *mtrace* in a multicast session would be 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 be able to access the data via the web site.

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, the Java applet would contact the instance of MHealth via a unicast TCP socket. Due to Java applet security restrictions, the MHealth application 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 was not scalable. As the quantity of data to be transferred 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. This approach has been abandoned for the time being.

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. Implementation of an image dump is currently being investigated.

5.2. Passive Mode MHealth

One interesting solution for the problem of of multiple MHealth sessions is to utilize *passive mtraces*. The results of most *mtraces* are multicast onto a well-known multicast address. A passive *mtrace* does not send an *mtrace Query*, but promiscuously listens for an *mtrace* which matches the user's 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 actively sending *mtrace* queries. In theory, this would create a single “leader” MHealth process that would send queries, and the other MHealth processes would be passive.

5.3. Total Mtrace Control and Partial Tree Tracing

An optimization of MHealth would be to integrate the *mtrace* functionality entirely into the Java application, rather than parsing the output of the existing *mtrace* tool. This would greatly enhance the reliability of interpreting the *mtrace* output and offer tighter control on the 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. Session Playback

One possible enhancement is the ability to play back a logged MHealth session, either in actual time or perhaps at some multiple of real-time. 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, almost like "time-lapse" photography. 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 of audio and video traffic. Multicast is shown to be growing in popularity, and Internet deployment efforts have been growing as well. A clear need for tools to monitor multicast traffic, diagnosis faults, and analyze traffic flow is identified. This need is often cited as one of the most significant barriers to widespread multicast adoption.

MHealth, a graphical, near real-time multicast monitoring tool has been developed. MHealth allows a user to view and collect data about the health and topology of a multicast tree. Despite concerns about MHealth's scalability and granularity of data collection, it has been shown to be a useful tool for collecting, processing, and archiving topology data.

The potential of multicast to change content distribution in the Internet and in traditional television, radio, and print media is enormous. The decade long effort to make multicast a ubiquitous feature is culminating in widespread adoption plans throughout the Internet infrastructure. MHealth is one of the first intuitive and easy-to-use visualization tools for group-wide multicast monitoring and debugging. Of critical importance is MHealth's ability to integrate data from multiple sources and provide a more comprehensive picture of multicast data flows. MHealth is probably the first of many tools that will be aimed at reducing the learning curve for managing multicast networks. Hopefully, MHealth is a next step in solving the "chicken-and-egg" problem of having the right tools before deploying multicast but also needing the right tools before multicast can be deployed.

References

[1] H. Eriksson, "The multicast backbone," *Communications of the ACM*, vol. 8, pp. 54–60, 1994.

[2] S. Casner, *Frequently Asked Questions(FAQ) on the Multicast Backbone(MBone)*. USC/ISI, December 1994. Available from ftp://ftp.isi.edu/mbone/faq.txt.

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

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

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

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

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

[8] D. Thaler, *Mstat*. Merit Network, Inc. and University of Michigan. http://www.merit.edu/net-research/mbone/mstat.html.

[9] D. Thaler and A. Adams, *Mrtree*. Merit Network, Inc. and University of Michigan. http: //www.merit.edu/ net-research/ mbone/ mrtree_man.html.

[10] D. Thaler, *Mview*. Merit Network, Inc. and University of Michigan. http://nic.merit.edu/∼mbone/mviewdoc/Welcome.html.

[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," Tech. Rep. RFC 896, Internet Engineering Task Force (IETF), January 1984.

[16] S. Deering, "Host extensions for ip multicasting," Tech. Rep. RFC 1112, Internet Engineering Task Force (IETF), August 1989.

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

[18] B. Cain, S. Deering, and A. Thyagarajan, "Internet group management protocol, version 3," Tech. Rep. draft-ietf-idmr-igmp-v3-*.txt, Internet Engineering Task Force (IETF), February 1999.