gTrace: Simple Mechanisms for Monitoring of Multicast Sessions

Gísli Hjálmtýsson, Ólafur Ragnar Helgason and Björn Brynjúlfsson

Networking Systems and Services Laboratory
Reykjavik University, Department of Computer Science
Ofanleiti 2, 103 Reykjavik, Iceland
{gisli, olafurr, bjorninn}@ru.is

Abstract. For multicast to be a viable as a dependable network service, mechanisms for monitoring and managing multicast flows must be developed. Significant prior work exists to use receiver observations to derive properties of multicast groups, including the underlying distribution topology and membership size. However, just as multicast is delegation of replication and distribution from the sender into the network, we contend that similar delegation of monitoring and management into the network is warranted and beneficial. gTrace is a set of mechanisms to monitor multicast distribution trees that operates on routers participating in the multicast distribution to obtain accurate observations throughout the tree in a scalable manner. In this paper we present the protocol and validate its benefits through analysis, simulation and experimentation.

1 Introduction

Multicast offers scalable delivery to large number of receivers by delegating work from senders to the network thereby conserving sender and network resources. However, multicast distribution introduces new and significant challenges. Some of these challenges arise from the inherent properties of multicast that invalidate principal assumptions of unicast Internet protocols. In particular, point-to-point protocols generally assume a closed feedback loop from the sender to the receiver. With multicast this is not the case. In IP multicast models the sender(s) are oblivious to the identity of the receivers and their individual capabilities and observed network performance.

In [1] Sarac et al survey tools for multicast monitoring, classifying them into debugging tools [2,3,4,5,6], management tools [7], and modeling tools [8,9]. Focusing on providing multicast over the MBone, these tools have significant drawbacks as mechanisms to provide consistent level of service quality for multicast sessions. Whereas the debugging tools aim at identifying reachability and loss problems in the MBone, the modeling tools aim at understanding long term behavior of the MBone as an infrastructure, rather than the performance of individual sessions. The management tools again focus on managing MBone connectivity and performance. Some of the problems with these tools include dependence on application layer data, end-to-end monitoring, implosion problems, limited responses, implosion and more. We conclude that all these tools have in common of being infrastructure monitoring tools and do not offer valuable mechanisms for service management.

To offer multicast as a dependable network service it is essential for service providers to be able to monitor service delivery. Service providers exploiting multicast, such as a TV station, or a gaming provider, need to be able to either monitor multicast delivery directly or contract the equivalent of an SLA assuring consistent service quality throughout the multicast tree. Knowing overall loss rate may enable the sender to adapt encoding to reduce bit rate. Identifying bottleneck links may allow congestion adaptation over the bottleneck link, such as local retransmission, transcoding, or selective discard of less important packets. Multicast protocols that implement stochastic feedback depend on knowing (an estimate of) the group size. Knowing group size may help service providers prioritize resource allocation; knowing properties of the distribution topology, including loss rates and bottleneck links helps in identifying and mitigating service delivery problems.

Of particular interest are recent ideas on autonomic networking where a root of a subtree may autonomically activate local retransmission, or may increase forwarding priority in reaction to high losses or long delays respectively. To facilitate, such local reaction, scalable approaches are needed to collect and propagate information such that the relevant information is available throughout the distribution tree.

In this paper we present gTrace, a protocol and mechanisms to monitor multicast distribution trees, combining a request/collect mechanism with local observations at the routers participating in the multicast. gTrace delegates the monitoring and data collection to the routers participating in the multicast to obtain accurate observations throughout the tree while incurring minimal overhead. In contrast to prior work, the protocol focuses on *individual distribution trees* and is applicable to any sparse mode multicast protocol and many application level multicast. Using gTrace all nodes of a given multicast distribution tree efficiently learn key properties of the tree, including topology information, loss rates, the group size, and tree height. While supporting an external observer, gTrace recursively propagates observations throughout the distribution tree. In particular each node in the distribution tree recursively collects (and has at its disposal) information about the downstream subtree. gTrace supports incremental deployment, and remains valuable even if only a fraction of the routers in the multicast distribution tree participate in the protocol.

The primary contribution of this paper is the specification of the gTrace protocol, and its validation using simulation and experimentation. After discussing related work in Section 2, each of the contribution is discussed in successive section. Section 3 describes the protocol, with Section 4 giving examples of usefulness by showing how some key properties can be obtained using gTrace. In Section 5 we evaluate the mechanisms through simulations to show tracking capabilities and to sensitivity analysis of the protocol. In Section 6 we discuss implementation and our experimentation with gTrace in our experimental multicast services for TV distribution and teleconferencing. We then conclude in Section 7.

2 Related Work

As discussed above most multicast monitoring tools have focused on monitoring the MBone infrastructure [2,3,4,5,6,7,8,9]. For example, Mtrace [2] supports tracing the path from a multicast receiver to the source. While potentially valuable, Mtrace is

the multicast equivalent of traceroute, does not scale, and is inappropriate for continuous monitoring of multicast. Same applies to MHealth [3], which uses RTCP information to collect the identity of session members before employing mtrace. Another debugging tool, RMPMon [10] relies on RTP monitoring with an SNMP based framework. Mantra[4] collects multicast routing information on the MBONE by querying multicast routers and collecting statistics. Tracetree [11] – a tool to discover the topology of a multicast distribution tree – suffers from implosion as each router in the multicast distribution tree sends a response to the original requestor. While gTrace provides a single set of mechanisms capable of providing all these debug functions (see Section 4), in contrast to the tools above, gTrace is lightweight enough to be used on a continuous basis and provides observations throughout the distribution tree.

MRM [5] generates traffic on designated test groups thus employing active probing to identify faults. HPMM [6] extends MRM to support passive monitoring of actual multicast traffic. HPMM arranges testers into a hierarchy to aggregate responses to faults in the multicast infrastructure to prevent implosion at the observer. gTrace employs similar while more general aggregation function to prevent implosion. Unlike [6] gTrace does obtain information from routers directly, providing more accurate information. Assuming comparable deployment of HPMM and gTrace, gTrace can be viewed as a generalization HPMM, while significantly more lightweight.

Another use of multicast and multicast monitoring is to use end-to-end measurement of multicast traffic as a method to infer internal network characteristics [12] and the logical topology of multicast trees [13]. Probes are sent from the source (root) to the multicast receivers (leaves). The end-to-end loss of probes is used to infer the logical multicast topology and the loss rates of the logical links in the topology. A significant drawback of these methods is that they only compute long term averages and do not adapt well to membership changes. In contrast, gTrace eliminates this guesswork as is obtains observations at routers participating in the multicast. Moreover, gTrace promptly tracks membership and topology changes, and can accurately pinpoint bottlenecks links and fate sharing subtrees.

Sharing some of the same characteristics, is the body of work estimating the size of a multicast group [14,15,16] where members of a multicast group periodically, or on demand, send a probabilistic acknowledgement to an observer. In contrast in gTrace, rapidly tracking the number of multicast participants in *every* subtree of the multicast topology is one of the elementary variables maintained.

3 The gTrace Protocol and Mechanisms

GTrace consists of mechanisms and a protocol to perform three main functions: 1) local maintenance and information collection at the multicast routers, 2) a distribution mechanism to propagate requests and information from a collection point to the multicast nodes of the topology, and 3) information gather to propagate information from leaves in the topology towards a collection point applying a summation function at each intermediate node.

To simplify the discussion, in this section we will assume that the collection point is the root of the multicast distribution tree being monitored. Each node in the tree apart from the root, has exactly one upstream neighbor and zero or more downstream neighbors. The gTrace state is identified by the root of the multicast tree and a group identifier. In single source multicast [17,18] this is the channel identifier. In PIM-SM and other shared tree approaches, the root of the multicast could be the rendezvous point, or the root of any of the source specific trees. The local state at each node mirrors that of multicast forwarding, having a downstream state for each downstream neighbor. However this state is control plane state and stored in regular memory and hence not size critical.

An external management system may employ gTrace. This is indeed how we see typical use by service providers, and how we have employed gTrace for service monitoring and management in our experimental services. However, this can be achieved by periodic report requests or by establishing an association between the external observer and a collection point within the distribution tree, typically the root of the multicast distribution tree.

Although in this paper we focus on gTrace for a single source distribution trees, we have experimented with extension to gTrace for more general topologies and overlays [19].

3.1 Local information collected at intermediate nodes.

The local information collection mechanism interfaces with the multicast topology management module to collect local multicast properties of interest. The topology management module is queried for active multicast flows and returns a list of active flows, each identified by the pair <*S,C*>.

For each multicast channel being monitored, gTrace maintains information about upstream neighbor, and set of next-hop downstream neighbors. For each downstream neighbor, the local state maintained by gTrace consists of the weight (w) i.e., total number of nodes, number of leaves (l), the height (h) of the subtree rooted by that neighbor and the true height (i) measured in physical hops based on TTL. Space complexity at each node is therefore four integer variables times the local fan-out of each monitored channel.

In addition gTrace can obtain from the multicast daemon the number of bytes (b) and packets (p) received on the flow <S,C>. We have defined an abstract interface through which the gTrace daemon queries the multicast topology module to export its state to the local information collection mechanism. This interface must be adapted to the particular multicast protocol in use.

3.2 gTrace protocol messages

We have defined three types of gTrace messages, a query, a report, and an update. A query message is used to subscribe to periodic updates of the query result. The report message is used to propagate information down the tree from the originator. The update is used by the gather mechanism as a response to a query.

The format of the gTrace message consists of a message type, type header, a data descriptor, and data values. The message type is one of query, report, or update, and is encoded as a single character. The type headers are described in the respective

subsections below. The message data descriptor is a null terminated string of characters, each character value corresponding to a particular variable of interest. The six basic local state variables – *w*, *l*, *h*, *i*, *b* and *p* – are encoded with the corresponding character. In addition the messages may have two fields for IP address and TTL value, encoded by *a* and *t* respectively. The values follow the string (at the next 32 bit boundary in our implementation) with the variables appearing in the same order as they appear within the string. All variables are encoded as 32 bit values, except the byte and packet counts which use 64 bits.

3.3 The distribution mechanism

The distribution mechanism is used to distribute queries and information from the collection point to the nodes of the topology. The originating node creates a distribution message and sends it on the multicast group. Distribution messages are sent with the router-alert (hop-by-hop) option down the multicast tree. Distribution messages are sent unreliably, but repeated three times on each link to overcome losses. [1]

Query subscriptions: The query message is used to subscribe to the continually updated computation of results specified by the message descriptor. The query message type header specifies an update period. The update period is given in milliseconds between updates. A null value in the update period cancels prior subscriptions if one exists. In addition a query always has the *a* and *t* variables specified. A gTrace node that receives a query sends the state values specified in the query message to the upstream gTrace node given as the value of the address field. Before forwarding the query message downstream, the node stamps its own address and the current TTL in the *a* and *t* fields of the message. Unless the update period is zero it then continues to send updates periodically, and stores the query locally. When a new branch joins the topology the query is forwarded on the new branch, extending the subscription to the newly joined downstream nodes.

Explicit reporting: The report message is used to propagate information down the tree, or report to an outside observer. The type header contains a report-to field containing an IP address, to which reports are to be sent, and a replace/trace flag, encoded as *r* or *x* respectively. If a trace is requested (i.e., r flag set), the processing of the report message is similar to the processing of a record route IP option, in that each node in the multicast distribution tree appends its values for the variables specified in the option header field before forwarding it on each of the output ports of the channel. Each subsequent hop in the path processes the message by appending one element to the array, containing the selected variables. When the report reaches a leaf (or the last hop gTrace router), a single result is sent to the report-to host.

If instead the replace flag is set, each node in the path replaces the values in the message with its own local values before forwarding the message downstream and to the report-to host. If no trace flags are set the original values are propagated to all nodes below the originator in the topology. For example, this is how the root may announce the group size to all receivers.

---

[1] If the three messages are sent sufficiently far apart, losses are independent. Even at relatively high loss rates, e.g. 8%, the probability of loosing all three is insignificant. As messages fit in a single packet, overhead is negligible.

3.4 The gather mechanism – processing subscriptions

The gather mechanism propagates information from the leaves of the topology towards the root by periodically sending updates upstream. At each node the gather involves two functions: a) Receiving and processing update messages from downstream, and b) preparing and sending of an update result upstream. The two functions are performed asynchronously.

The values in a gather message received from downstream on a given channel is simply stored as part of the channel gather state, and override any previous updates from the same downstream node. The message format of the update is the same as of the originating query subscription.

Periodically, for each multicast flow, each router computes a new local state and sends an update to its upstream router. The new local state is computed from local observations and from the state updates from downstream neighbors of the multicast topology. The number of bytes and packets, b, and p, received on the multicast from source S are observed locally. The remaining values, w, l, h, and i are computed by applying an appropriate summation function respectively as

$$w = 1 + \sum_{j \in out(v)} w_j \qquad l = \sum_{j \in out(v)} l_j$$

$$h = 1 + \max_{j \in out(v)} \{h_j\} \quad i = \max_{j \in out(v)} \{i_j + \Delta(TTL)\}$$

where *out*(v) denotes the set of output ports for the multicast, and Δ(TTL) is the difference in TTL of the IP header of the message and the t value in the message.

For each active flow, gTrace includes a timer field that denotes the time when the next local update is to be sent upstream, and for the output ports denotes the time when a state update was last received. If the state update is managed as part of refreshing the multicast forwarding state these timers are superfluous, and omitted.

To economize on message processing, the state update may be sent as part of the message(s) for refreshing the multicast forwarding state. Alternatively, the state updates of multiple multicast groups may be combined and reported in a single message. For a particular flow the updates are sent asynchronously and independently. The update messages are sent upstream on the incoming port of the multicast flow.

The gather process starts at the leaves or at last-hop routers. If the end-systems do not participate in gTrace the last-hop router sets $h = i = 1$ and may attempt to estimate the appropriate values to w and l, depending on the information that it has available from the local collection mechanism (which may be protocol dependent). Without superior information, the router sets l to its local outdegree (for the group), and w to $l + 1$. To gracefully leave a session when a node leaves the multicast group the node sends a last gTrace message with $w = l = h = i = 0$.

An important tradeoff in the realization of the gather mechanism is the length of the update period. A short update period gives better state estimates at the increased overhead cost of bandwidth and processing at the nodes. The update period can be adjusted by the collection point by resending the query with a new update period. In Section 5 we evaluate the gTrace gather mechanism through simulation to evaluate the effect of the update period on the estimation lag and error.

All gTrace messages are sent unreliably. However since an upstream node simply maintains the last received update as its current best estimate (as long as the multicast

branch is active), under reasonable loss rates, losses merely delay the propagation of information updates. Of course this contributes to errors in the estimates of the various values, as discussed in the next two sections.

3.5 Incremental deployment

Initially during the deployment of the gTrace mechanisms, one cannot assume that all multicast routers participate in the protocol. A router that does not implement the gTrace protocol will simply forward report messages without processing. This is not a major issue as the distribution messages are forwarded on the multicast group, and thus all routers and receivers will receive the message. Updates are sent point-to-point to the address of the closest upstream gTrace router learned from the query message. In fact, by comparing the TTL value of the message to that of the current TTL in the IP header of the packet containing the query message the downstream gTrace nodes can determine how many non-cooperating routers there are between it and the next upstream gTrace enabled router.

4 Computing Multicast Attributes

In this section we describe how to use gTrace to compute a number of interesting properties of multicast. We consider two types of applications, group centric, and network centric. The former is used by application level protocols or a group manager to adapt and optimize the behavior of the application level protocols. The latter is used by network and service management systems to monitor and manage resources allocated to the multicast groups, and apply network policies for resource sharing. The application sets the gTrace options depending on the needs of the group. Note that although the data collection is coordinated, it is still an estimate due to staleness caused by protocol delays and the group dynamics.

Estimating group size: The leaf count *l,* at the collection point (root of the distribution tree) is the current estimate of the group size. More generally, the leaf count at any intermediate node is the down-stream group size. For some applications (e.g. applications using RTP) it is important for the leafs to know the group size. This information can be propagated to the receivers by sending a gTrace report from the multicast root. The source records its own information, source address, weight, height and number of bytes sent. Without any flags set the intermediate routers simply forward the message on the multicast tree. On receipt, receivers learn the current estimated group size.

Identifying and locating bottleneck links: Since number of packets received is collected and exchanged, gTrace can be used to identify a bottleneck link in a multicast distribution tree. Each node periodically transmits upstream the number of packets it receives on a flow. By comparing the local receive rate to the receive rate on a given downstream branch (from the gather update), a node can determine if the losses on the downstream link are "excessive." This information could be used to activate local retransmissions on that particular link or employ other forms of local congestion adaptation [20].

A more thorough analysis of losses and in fact topology can be obtained by using a report message with the record flag to propagate and collect the addresses and

received packets of every node at a service management host (specified in the report-to field) for analysis.

Determining the number of receivers sharing a bottleneck link: Using the above and including the weight, and/or leafs in the subscription the nodes above the bottleneck knows the number of node and/or leafs below that link. A similarly augmented report message, i.e. requesting weight and leafs, can be employed to deliver that information to an external observer.

Topology discovery: We can use gTrace to discover the topology of the multicast in a number of ways. Sending a report with the *trace* flag set and address field selected, propagates the traceroute from the root to each node in the tree. By specifying a *report-to* this information will be sent from each leaf to an external observer. To appreciate the underlying topology, including the TTL value as well is valuable. Using the same with the *replace* flag implements essentially the same semantics as [11].

5 Evaluation

Although the protocol is based on direct measurements the aggregate values seen at the collection point are the result of a distributed computation by the gather mechanism with information exchanged over an unreliable channel. Consequently the values at the root are (random) estimates of the true values. There are three sources of errors: staleness – since the values are observed at the root some time after the observations were collected; inconsistent observations – since the aggregate is computed from observations made asynchronously; and losses of update messages. In this section we use simulations to examine the dynamic behavior of gTrace, in particular its ability to track dynamic changes in topologies. For this purpose we have built a simulation model of gTrace mechanisms using the ns2 network simulator. Analysis of the steady state distributions is found in [21].

5.1 The basic model

In each simulation run we construct a multicast topology and a finite set of potential receivers. Receivers are either busy, or idle, becoming busy by joining a multicast group and idle when leaving the group. We use a two phase model with exponential holding times to model client activity.

We have used both regular topologies (binary- and k-ary trees) and random topologies. The random distribution trees are constructed in a recursive manner as follows: Each node has some probability α_{leaf} of being a leaf. If not a leaf, the fan-out of the node is determined as a random variable. We use a geometric distribution with mean 1.8 to generate sparse trees, and a normal distribution with mean 2.5 to generate more dense trees. To limit the size of the tree we limit the maximum height of the tree.

In our simulations we assume an end-to-end loss rate of approximately 4% from the deepest leaf to root. Each link has the same loss rate and losses are independent and identically distributed. Ignoring the processing overhead at nodes we use the same propagation delay for all links, 20 ms. While this is rather long, it is still an order of magnitude smaller than the smallest update period we consider.

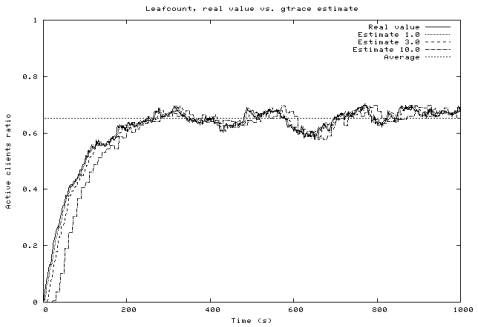

Figure 1: Dynamic tracking of gTrace using update period of 1, 3 and 10 seconds, over a full binary tree of height 8.

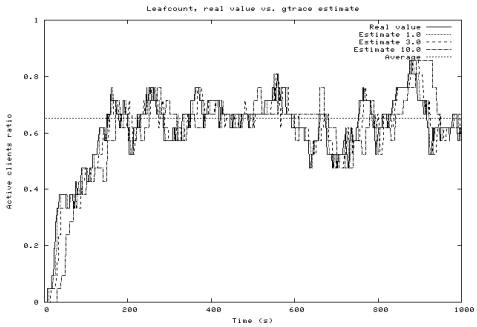

Figure 2: Dynamic tracking of gTrace over a random tree of height 10 with 21 potential receivers.

5.2 The ability to track dynamic changes in topology

To evaluate how gTrace handles dynamic topology we use the model described above with expected idle time of 100 sec, and expected active time of 200 sec. Each receiver joins and leaves according to this process repeatedly throughout the entire simulation.

Figure 1 shows a simulation run for a full binary tree of height 8 (thus having 256 leafs). The figure plots four curves, the true number of active receivers and the corresponding gtrace estimate (leafcount) at the root of the topology for three different update periods: 1, 3 and 10 seconds. To facilitate comparison with other figures, the y-axis is normalized to show the fraction of potential receivers active. As is evident form the figure, gTrace tracks the actual value excellently even with the 10 second update periods.

Figure 2 shows the same for a random tree of maximum height 10 having 72 nodes, thereof 21 leafs, created using $\alpha_{leaf} = 0.5$ and a geometrically distributed fan-out. The significantly fewer leaves result in substantially less activity, accounting for the steps in the curves. Again, we see that gTrace accurately tracks the actual

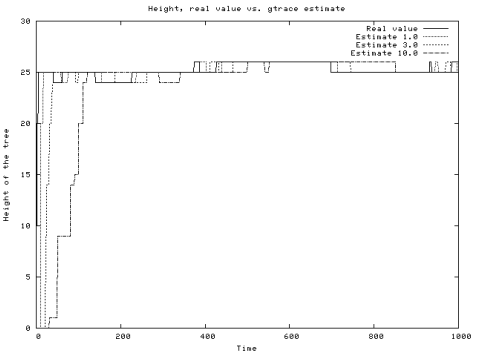

Figure 3: Dynamic tracking of height in gTrace over a random tree of height 26 with 150 potential receivers.

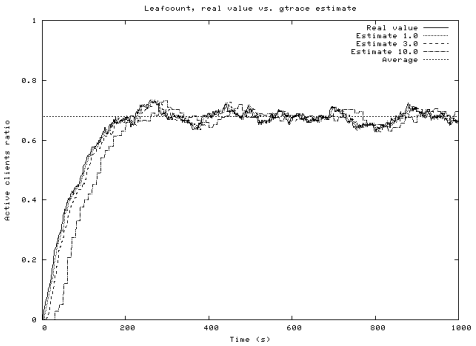

Figure 4: Dynamic tracking of over a random tree of height 8 with 279 potential receivers.

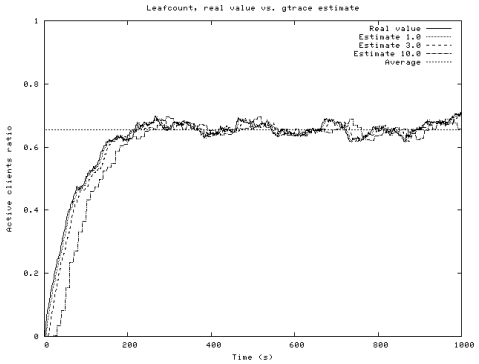

Figure 5: Dynamic tracking of gTrace over a full binary tree of height 9 with 512 potential receivers.

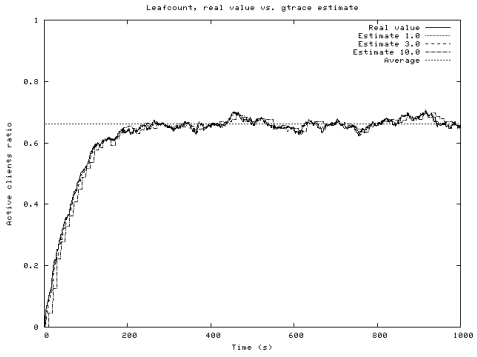

Figure 6: Dynamic tracking of gTrace over a full 8-ary tree of height 3 with 512 potential receivers.

membership.

gTrace ability to track the height of a tree is evident on Figure 3 which shows a simulation run for a random tree of having 447 nodes, thereof 150 leafs, the furthest leaf having height 26. The tree was created using αleaf = 0.3 and a geometrically distributed fan-out. The client behavior is the as before except using expected idle time of 200 sec, and expected active time of 100 sec to get more dynamic in the height. While the significant height results in appreciable lag for the 10 second update time, the figure shows that gTrace accurately tracks the height.

5.3 Insensitivity to the underlying topology

Figure 4 shows again the dynamic tracking of gTrace for a denser tree topology. The tree was created with $\alpha_{leaf} = 0.5$ and a normally distributed fan-out with expected value of 3, and maximum height set to 8, yielding a tree with comparably many receivers as the full binary tree in Figure 1, namely having 279 leafs out of a total of 690 nodes. Comparing Figure 1 and Figure 4 we see how gTrace is indifferent to the underlying topology as the figures are almost identical.

5.4 Effect of tree height on estimates

Figure 5 and 6 show the effect of height on the lag in information updates, depicting two scenarios with equal number of clients but with different height. For Figure 5 the tree topology is a full binary tree with 512 receivers and thus height of 9. Figure 6 shows a full 8-ary tree topology with 512 clients and a height of 3. For the binary tree and update period of 10 seconds the lag is clearly visible while the difference for the 8-ary tree is negligible.

6 Implementation and Experimentation

We have prototyped and experimented with gTrace for monitoring and managing multicast services, such as TV distribution and teleconferencing that we are

experimenting with over the Internet. Our multicast services are offered over our own novel single source multicast protocol, SLIM [18]. Initiated from our service management center we use the gTrace mechanisms to collect statistics about use and to monitor changes in membership and topology.

Local information collection at intermediate nodes: A gTrace local collection object communicates with the topology management daemon of SLIM to collect the local state information. We have implemented the abstract interface using shared memory minimizing IPC overheads.

The distribution and gather mechanisms: On receiving a query the daemon sets up the appropriate measurement state and an update timer value as specified by the message descriptor. When a new branch joins the multicast tree the daemon forwards the last query out on the branch. When a report message arrives the daemon serves the report by appending appropriate values, forwards the message and sends a copy to a report-to host if applicable. A gather message from a downstream node triggers the daemon to update its estimates for that particular sub-tree. On update timer expiration for a particular group the local state is collected from the SLIM daemon as well as the current estimates from downstream nodes and are then sent to the upstream router.

Experimentation with multicast services: We have been offering multiple multicast based services based on our SLIM protocol, including TV distribution, radio and conferencing on experimental basis for over two years. In particular we offer popular TV channels not available in remote areas in Iceland, at times received by hundreds of concurrent users outside of our campus. gTrace is proving lightweight and efficient in supporting the monitoring of multicast in this setting.

7 Conclusion

In this paper we have introduced gTrace, a control plane mechanism for measuring and monitoring multicast topologies. The gTrace control plane protocol consists of a distribution and gather mechanism, and three types of messages. Collectively the protocol allows for observations from *inside* of the multicast topology to be combined in a scalable manner to compute a flora of interesting attributes of multicast distribution trees. Our simulations show that gTrace is able to track key attributes accurately and is relatively insensitive to topology and dynamic member changes. This we have indeed verified in our own use of the protocol in our experimental multicast service for TV distribution and teleconferencing.

8 References

[1] K. Sarac and K. Almeroth, "Supporting Multicast Deployment Efforts: A Survey of Tools for Multicast Monitoring", Journal of High Speed Networking - Special Issue on Management of Multimedia Networking, vol. 9, num. 3/4, pp. 191-211, March 2001.

[2] Bill Fenner and Steve Casner, “A traceroute facility for IP multicast”. Internet Draft, work in progress, Internet Engineering Task Force, July 2000.

[3] D. Makofske and K. Almeroth, "MHealth: A Real-Time Multicast Tree Visualization and Monitoring Tool", Network and Operating System Support for Digital Audio and Video (NOSSDAV '99) , Basking Ridge New Jersey, USA, June 1999.

[4] P. Rajvaidya and K. Almeroth, “A Router-Based Technique for Monitoring the Next-Generation of Internet Multicast Protocols”, International Conference on Parallel Processing (ICPP), Valencia, Spain, September 2001.
[5] K. Almeroth and L. Wei, “Multicast Reachability Monitor”, IETF Internet Draft, work in progress, July 2000, draft-ietf-mboned-mrm-01.txt
[6] J. and B. N. Levine, “A Hierarchical Multicast Monitoring Scheme” in International Workshop on Networked Group Communication (NGC), Palo Alto, California, November 2000.
[7] D. Thaler, @Globally distributed troubleshooting (GDT): Protocol specification,” IETF draft, draft-thaler-gdt-*.txt, January 1997.
[8] K. Almeroth and M. Ammar, “Multicast group behavior in the Internet’s multicast backbone (Mbone),” in IEEE Communications, vol. 35, pp. 224-229, June 1997.
[9] M. Yajnik, J. Kurose, and D. Towsley, “Packet loss correlation in the MBone multicast network,” in IEEE Global Internet Conference, London, UK, Nov. 1996
[10] J. Chesterfield, L. Breslau, W. Fenner, “Remote Multicast Monitoring Using the RTP MIB”, in the proceedings of MMNS '02, Santa Barbara, 2002.
[11] K. Sarac and K. Almeroth, "Tracetree: A Scalable Mechanism to Discover Multicast Tree Topologies in the Network", accepted for publication in IEEE/ACM Transactions on Networking.
[12] N.G. Duffield, F. Lo Presti, “Multicast Inference of Packet Delay Variance at Interior Network Links”, in Proc. IEEE Infocom 2000, Tel Aviv, Israel, March 26-30, 2000
[13] N.G. Duffield, J. Horowitz, F. Lo Presti, D. Towsley, “Multicast Topology Inference from Measured End-to-End Loss”, IEEE Trans. on Information Theory, vol. 48, pp. 26–45, 2002.
[14] J.-C. Bolot, T. Turletti, and I. Wakeman, “Scalable feedback control for multicast video distribution in the Internet”. In Proc. of ACM SIGCOMM’94, London, UK, pages 58–67, September 1994.
[15] S. Alouf, E. Altman, P. Nain, "Optimal on-line estimation of the size of a dynamic multicast group". In Proc. IEEE Infocom 2002, New York, USA, June 2002.
[16] T. Friedman, D. Towsley, "Multicast Session Membership Size Estimation". In Proc. IEEE Infocom'99, New York, USA, March 1999.
[17] Holbrook H., and D.R. Cheriton, IP multicast channels: EXPRESS support for large-scale single-source applications, in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 1999.
[18] G. Hjálmtýsson, B. Brynjúlfsson and Ó. R. Helgason, "Self-configuring Lightweight Internet Multicast", in proceedings of IEEE SMC 2004, Hague, Netherlands, October 2004.
[19] Gísli Hjálmtýsson, Ólafur Ragnar Helgason and Björn Brynjúlfsson, "Simple Active Mechanisms for Measuring and Monitoring Service Level Topologies", in proceedings of IWAN 2004, Lawrence Kansas, USA, October 2004.
[20] Gísli Hjálmtýsson and Samrat Bhattacharjee, “Control on Demand – An Efficient Approach to Router Programmability,” in IEEE JSAC, Vol. 17, No. 9, September 1999, pp. 1549-1562.
[21] Gísli Hjálmtýsson, Ólafur Ragnar Helgason and Björn Brynjúlfsson, "Analysis of gTrace", Reykjavik University Technical Report, NL200404, October 2004.