Scalable Techniques for Discovering Multicast Tree Topology

Kamil Saraç
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
ksarac@cs.ucsb.edu

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

ABSTRACT

The IP multicast infrastructure has transitioned to a topology that now supports hierarchical routing. Multicast network monitoring and management have become key requirements necessary for providing robust multicast operation. Monitoring services help to identify potential problems such as protocol shortcomings, implementation bugs or configuration errors. This type of monitoring often requires knowing the multicast tree topology. In this paper, we present a new approach, called tracetree, to discover tree topology in the source-to-receiver(s) direction using network forwarding state. We start with an overview of the problem. Then, we describe tracetree functionality including its request forwarding and response collection mechanisms. Next, we discuss a number of functional issues related to tracetree. Finally, we evaluate our technique by comparing it to a number of alternative approaches. We believe that our technique provides a scalable way of discovering a multicast tree's topology in realtime while requiring only marginal additional router functionality.

1. INTRODUCTION

The IP multicast infrastructure has transitioned to a topology that now supports hierarchical routing. Today, most router vendors support native multicast routing and Internet Service Providers (ISPs) are starting to deploy multicast in their networks. In this hierarchical topology, ISPs run potentially different multicast routing protocols within their domain and use a particular set of protocols to provide inter-domain multicast support[1].

Commercial ISPs have been experiencing difficulties in deploying multicast in their networks[2]. These difficulties are mainly related to the instability and complexity of multicast routing. It is believed that the availability of good management tools has become a crucial roadblock to successful multicast deployment.

Discovering multicast tree topology is an important component of multicast network management[3]. Problems due to routing protocol limitations, multicast network misconfigurations, or routing policy decisions can easily affect multicast forwarding paths. This may prevent some receivers from successfully receiving session data or may cause some portion of the network to continue carrying multicast data unnecessarily. Tree topology discovery is useful for both end users and network managers. End users can use topology information and traffic flow to determine whether there is activity in a group, or, if there is a problem, where to direct an inquiry. Network managers can use topology information as the basis of group monitoring, or can use it to identify potential multicast forwarding problems.

Multicast tree topology discovery is a more difficult challenge than unicast path discovery. The unicast path between a local site and a remote site can be discovered by using routing information in the local-to-remote direction (only). In multicast, routing state in the network is used to create a multicast tree between receiver sites and the session source site[1]. Routers on the tree create *multicast forwarding state* for the group. The forwarding state at each router contains the interface that the multicast data from the session source is expected to arrive on and the interface(s) on which the multicast data is to be forwarded. Therefore, in multicast, the tree topology can be discovered in two different directions:

- **The receiver(s)-to-source direction:** Multicast *routing* information is used to discover the tree topology. First, the multicast path from each receiver to the source site is traced. Then, the collected information

NOSSDAV'01, June 25-26, 2001, Port Jefferson, New York, USA.

[1]In general, these trees can be shared trees or source specific trees. For simplicity, we only consider source specific multicast trees but similar arguments apply to shared trees.

is used to build a tree[4, 5]. This approach requires knowing the identities of all session receivers.

- **The source-to-receiver(s) direction:** Multicast *forwarding* state is used to discover the tree topology. Topology discovery starts from the root of the multicast tree and progresses towards the receivers. This approach does not require knowing the identities of session receivers.

In this paper, we attempt to develop mechanisms that would be used to discover a multicast tree's topology in the source-to-receiver(s) direction using multicast forwarding state *only*. Our approach does not require knowing the network topology and does not require access to Simple Network Management Protocol (SNMP) information in the routers. We call our approach *tracetree*.

Topology discovery in the source-to-receiver(s) direction using forwarding state has been considered impractical due to scoping difficulties and its many-to-one communication requirements. In our work, we explore these issues. We try to identify the potential problems and propose solutions. We see *tracetree* as orthogonal to the *mtrace* tool and believe that the existing *mtrace* code in routers can be extended to support tree topology discovery functionality.

The remainder of the paper is organized as follows. The next section is on related work. Section 3 presents our approach for discovering tree topology information. Section 4 discusses issues related to our topology discovery mechanism. Section 5 addresses preliminary evaluations for our techniques. The paper is concluded in Section 6.

2. RELATED WORK

There are a number of existing multicast management tools that collect multicast tree topology for monitoring purposes. These tools can be grouped into SNMP-based tools, i.e. *mrtree* and *mmon*, and non-SNMP-based tools, i.e. *mrdebug* and *mhealth*. In this section, we briefly describe *mmon* and *mhealth* as example tools in these groups.

Mmon is an SNMP-based software tool developed at HP Labs[6]. It was developed primarily for managers of IP multicast networks. *Mmon* uses the existing network topology information and a number of multicast related Management Information Base (MIB) tables in the routers to discover a multicast tree's topology. *Mmon* is well-suited for use by Network Operation Center (NOC) personnel within an administrative domain. But it is not usable by end users or for inter-domain monitoring. This is because ISPs are not likely to give detailed access to SNMP information in their network devices to either end users or network managers in other domains.

Mhealth, the Multicast Health Monitor, is a non-SNMP-based tool that is developed to discover tree topology[4]. *Mhealth* works in the receiver(s)-to-source direction. It uses *mtrace* and the Realtime Transport Control Protocol (RTCP) to discover tree topology. *Mtrace* is a multicast version of the *traceroute* utility[5]. It returns a multicast path between a given group receiver and the group source. The trace starts at the receiver site and works in the reverse direction towards the source site. Tree topology discovery using *mtrace* requires knowing the identity of session members. *Mhealth* uses RTCP reports to collect this information. RTCP is defined as part of the Realtime Transport Protocol (RTP)[7]. RTCP specifies periodic transmission of control packets by all group members to all other group members. These control packets are sent to the session group address and are expected to be received by all session members.

An advantage of *mhealth* over the SNMP-based tools is that it is a user level tool and it works on an inter-domain scale. On the other hand, *mhealth* depends on RTCP data which is unreliable. In addition, emerging multicast routing protocols e.g. Source Specific Multicast (SSM)[8], break the RTCP mechanism by allowing only the session source to transmit data to the group multicast address. Finally, execution of *mtraces* for each receiver is not scalable[4].

3. *TRACETREE* FUNCTIONALITY

As mentioned above, our general approach is to discover tree topology in the source-to-receiver(s) direction. This approach depends on the ability to send *tracetree* requests to a multicast group address and have routers recognize and respond to these requests. Since the request packets are destined to multicast group address, we need a mechanism for on-tree routers to distinguish *tracetree* request packets from the regular data packets in the group. For this we use the IP Router Alert option[9] in the *tracetree* request packets. In general, the Router Alert option is used to alert transit routers to more closely examine the contents of an IP packet. In our case, this option enables routers to recognize the *tracetree* request packets and process them. In the example in Figure 1, a third party querier, Q, is interested in discovering the multicast tree topology for group (S, G). Q sends a *tracetree query* message to the first hop router, A, at the session source site, S[2]. On receiving this query message, A changes it to a *tracetree request* packet and forwards it to the session multicast address, G. In forwarding the request, A spoofs the IP address of session source S and uses it as the IP source for the request packets. This prevents on-tree routers from creating new forwarding states for

[2]Locating first hop router A is discussed in Section 4.1

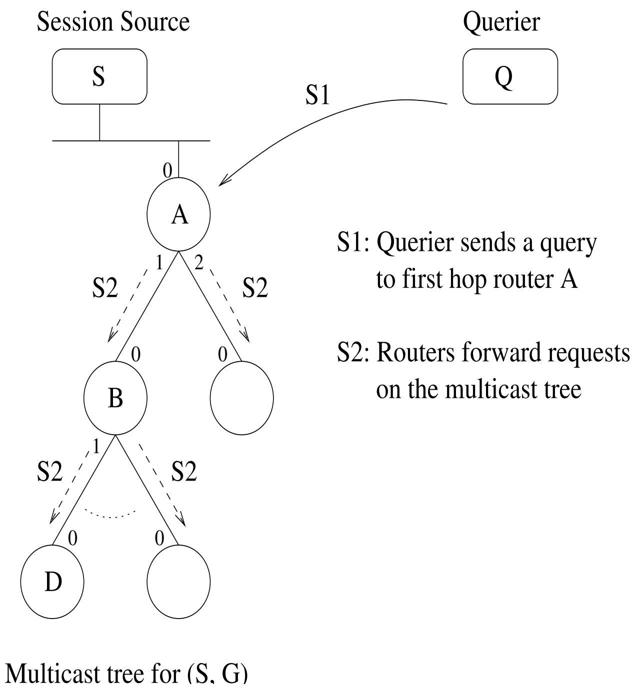

Figure 1: Overview of topology collection.

(A, G) which may not even be possible for some multicast routing protocols. In addition, A puts its own IP address as *request forwarder* into the *tracetree* protocol header. Lastly, A sets the IP Router Alert option so that routers receiving this request packet recognize and process it. Each on-tree router, upon receiving a request packet, replaces the *request forwarder* field in the incoming request packet with its own IP address and forwards it down the tree as long as the scope of the packet allows.

The next step is to collect responses from the routers. In the rest of this section, we introduce two different approaches for collecting this information: *message-based* response collection and *stateful* response collection techniques.

3.1 Message-Based Response Collection

The *message-based* response collection technique requires routers to participate in the response collection task by directly sending their responses to the querier site. This technique requires relatively low processing overhead in the routers. However, depending on the size of the multicast tree, it may create significant messaging overhead and require significant processing overhead at the querier site. Therefore, it is necessary to control the *response arrival rate* at the querier site. One way to control this rate is to use a randomized delayed response mechanism. The random delay interval is determined by the querier based on its estimate of the tree size. While the delayed response mechanism provides scalability for relatively small multicast trees, for large trees, either the response rate becomes very high or the maximum delay time becomes very large. In addition, it may not always be possible to estimate the size of a multicast tree correctly.

An alternative collection method is to divide the response collection process into *rounds*. This technique provides better scalability. In this approach, Q informs A about the number of responses, $n_{response}$, that it wants to receive in a round. A uses this information to compute a scope for the request packets. The scope calculation is based on a *modified* Time-to-Live (TTL) computation. In computing the TTL value for the request packets, routers use the number of downstream links they have on the multicast tree. In Figure 1, assume that an on-tree router, B, receives a *tracetree request* packet from its upstream neighbor on the tree. In addition, assume that B has *num_children* outgoing links on the multicast tree. Let the current IP TTL value in the incoming request packet be $TTL_{current}$. B computes a new TTL value, TTL_{new}, for request packets that it forwards down the tree as

$$TTL_{new} = \frac{TTL_{current} - 1}{num\_children}. \tag{1}$$

There are several issues associated with making sure that roughly $n_{response}$ messages are returned per round. These issues are discussed in Section 4. After the querier is done with a round, it sends additional *tracetree queries* to appropriate subtree routers to continue topology discovery.

3.2 Stateful Response Collection

The *stateful* response collection technique requires routers to more actively participate in the response collection mechanism. The message-based response collection method relies heavily on messaging to collect topology information. Our main focus in this second approach is to reduce this overhead and move the complexity into the routers. In this approach, routers send their responses back toward the source instead of to the querier. Routers on the tree keep temporary state and store partial topology information. In forwarding requests, routers start a timer and wait for a period of time to receive responses from downstream links. All the incoming responses are temporarily stored in a *reserved memory location*. At the end of a timeout period or after the routers receive a response from all downstream links, they append the received response messages to their own response block and send them back to the upstream router. In the case of the first hop router, it sends the accumulated responses back to the querier.

In the stateful response collection method, the scope of request packets is controlled by using the size/memory limit in the incoming request packet or in the router. Upon receiving a query message, the first hop router, A, determines the amount of storage space that it can reserve for this query. Then, A informs the downstream routers about this size limit in its request packet. Similarly, on-tree routers compare the expected size value with the amount of memory that they can reserve for the request. If the scope of a request message is not large enough to forward, then the router immediately sends its response back to the upstream router leaving topology discovery for this subtree to a future round.

In this approach, routers keep timers while waiting for response messages from their downstream neighbors. In order to prevent premature timeout expirations, routers should select the timeout values carefully and should inform their downstream routers about it. Note that using short timeout values may cause premature scope expirations. That is, even though a request packet has large enough size limit to be forwarded down, a short timeout duration set by an upstream router may prevent it from being forwarded down in the tree.

4. RELEVANT *TRACETREE* TOPOLOGY DISCOVERY ISSUES

4.1 Finding the First Hop Router

Before sending a *tracetree* query message, the querier needs to know the IP address of the first hop router. We assume that the querier obtains this information externally and uses it to initiate the topology collection process. For a network administrator or a local querier, this information is easy to determine. On the other hand, for distant third party queriers this information may be difficult to obtain. One option might be to run a group specific *mtrace* towards the session source.

4.2 Existence of Non-compliant Routers

The existence of non-compliant routers may cause uncontrolled replication of request messages. If not controlled, this replication may cause potential scalability problems in the response collection mechanism. For example, in case of the message-based response collection technique, non-compliant routers would use the standard TTL decrement mechanism when forwarding request packets. This would interfere with the modified-TTL-based scope control mechanism for requests. In the case of stateful response collection, non-compliant routers would not participate in response collection at all.

In order to detect the existence of non-compliant routers, routers use a technique based on the duplication of the TTL value in the request packets. On forwarding a request packet, routers copy the IP TTL value into the *tracetree* protocol header. This new value is called TTL_{tt}. Upon receiving a request, routers compare the values in the IP TTL and the TTL_{tt} fields. If these two values are equal to each other, it implies that the upstream router forwarding the request is a compliant router. If the values are different, the difference gives the number of non-compliant routers since the last compliant router. After detecting the existence of non-compliant routers, a decision on what to be done must be made. This depends on the response collection mechanism being used.

Message-based response collection: The first compliant router after the non-compliant router(s) uses an *average branching factor (ABF)* to re-compute the TTL value of the incoming request packet. The assumption here is that, on average, each router in the network has ABF outgoing interfaces for the multicast tree. By using ABF and TTL_{tt}, routers can compute a modified TTL_{IP} value as

$$TTL_{IP}^{'} = \frac{TTL_{tt}}{(ABF)^{(num\_level)}} \qquad (2)$$

where $num\_level = TTL_{tt} - TTL_{IP}$.

In a recent study, Chalmers and Almeroth studied branching characteristics of global multicast trees[10]. In this study, they identified the ABF for internal multicast routers to be approximately 1.57 (depending on the size of the tree) with a standard deviation of 1.27. Based on these values, 3 can be used as a conservative approximation for ABF. After updating the TTL value of the incoming request packet to $TTL_{IP}^{'}$, routers may take one of a number of possible actions: (1) continue normal processing, (2) send a response to the querier if $TTL_{IP}^{'}$ is larger than one, but do not forward the request, and (3) do not respond and do not forward the request. These alternatives have different implications in terms of scalability and completeness.

Similar to the case with non-compliant routers, the existence of multi-access links between on-tree routers causes irregularities in scope calculations. When a router has an outgoing multi-access link that is on the multicast tree, it uses the above equation to calculate the TTL value.

Stateful response collection: For the stateful response collection mechanism, non-compliant routers may cause upstream routers to receive multiple responses from a downstream link. In this situation, routers can process these multiple responses as long as they do not overflow size/memory limits.

4.3 Handling the Loss of Response Messages

Depending on the response collection mechanism, loss of response messages has different effects on the topology discovery process. In the message-based response collection technique, loss of response messages may cause gaps in the tree topology. Missing links may occur if a response message from an on-tree router is lost, but the response message coming from its downstream neighbor arrives at the querier site. For example, assume that the response of B in Figure 1 is lost but that of D reaches Q. In such a situation, Q can send a new *tracetree query* message to A and asks it to create and forward a request packet only on its interface IP_{A_1} with a TTL value of one. This causes B to receive and respond to this request again. In general, by using additional control information in the request packets, the querier can better isolate where missing topology information belongs.

For the stateful topology collection mechanism, loss of a response message is detected by the upstream router and the topology discovery for this subtree is left to a future round.

4.4 *Tracetree* Security

There are some potential security risks regarding *tracetree* functionality. First, using the IP Router Alert option puts extra workload on the routers. In addition, a large number of *tracetree requests* can cause a heavy workload on routers. To solve this problem, routers can use a request processing rate limit to control overhead. Similarly, in the case of the stateful response collection approach, routers can limit the amount of memory dedicated to storing tree topology state.

Second, in the message-based response collection mechanism, a malicious user can spoof the IP address of a third party site and identify it as a querier causing routers to send a potentially large number of responses to this site. In order to make this type of attacks difficult, routers can use a three-way handshake mechanism in communicating *tracetree* queries. Furthermore, using a request processing rate limit reduces the effect of such attacks. Finally, the IP TTL field is an 8-bit field which limits the number of responses to 255.

A final security issue is the possibility of injecting fake *request* packets into the tree. We believe that such an attack would be difficult to accomplish due to the multicast forwarding rules, specifically the Reverse Path Forwarding (RPF) check.

5. EVALUATIONS

We evaluate the performance of *tracetree* by comparing it to a number of alternatives namely *plain mtrace*, *hop-by-hop mtrace*, and SNMP-based approaches. In *plain mtrace*, each router appends its response block to the request packet and forwards it to the upstream router. When the request packet reaches the first hop router at the source site, it contains the whole path information. If *plain mtrace* is not successful, it switches to a hop-by-hop mode. *Hop-by-hop mtrace* works similar to *traceroute*. Requests start with a TTL of one which is incremented after successfully receiving a response from a router. In the SNMP-based approach, the querier uses a network topology map to identify next-hop routers and sends its queries accordingly. As we mentioned before, *mtrace*-based approaches require knowing the identities of session receivers and SNMP-based approach requires administrative privileges to access and use router MIB information.

The metrics we use for comparisons are (1) the processing overhead put on the routers, (2) the messaging overhead on the management site, and (3) the topology discovery time, i.e. the time needed to discover the topology. Figure 2 shows the generic operation of each technique and the amount of router overhead (in terms of number of visits to routers) and the amount of messaging overhead on the management site (in terms of number of messages reaching the management site). The numbers within routers show the number of visits to the routers and the numbers associated with links show when more than one message is sent to the management site.

For our evaluations, we simulate the topology collection techniques using the *ns-2*[3] network simulator. In our simulations we generate multicast tree topologies using a realistic data set collected by Chalmers and Almeroth. In the rest of this section, we first introduce our data set and then discuss the results of our comparisons in more detail.

5.1 Data Set

In our simulations we use the *mwalk*[10] data set to generate multicast tree topologies. This data set is generated by tracing multicast paths between a local source site and a large number of external IP addresses. The external IP addresses have been known to have participated in multicast groups at some point over the last 10 years. These IP addresses were collected by the *mlisten*[11] tool which joins multicast groups advertised on the well-known Session Announcement Protocol (SAP) address (SAP.MCAST.NET) and collects RTCP packets from the group members. In this way, the *mwalk* authors were able to trace nearly 2000 multicast paths. These traces were conducted within the last year and they represent the multicast infrastructure topology at that time.

[3]See http://www-mash.cs.berkeley.edu/ns/ns.html.

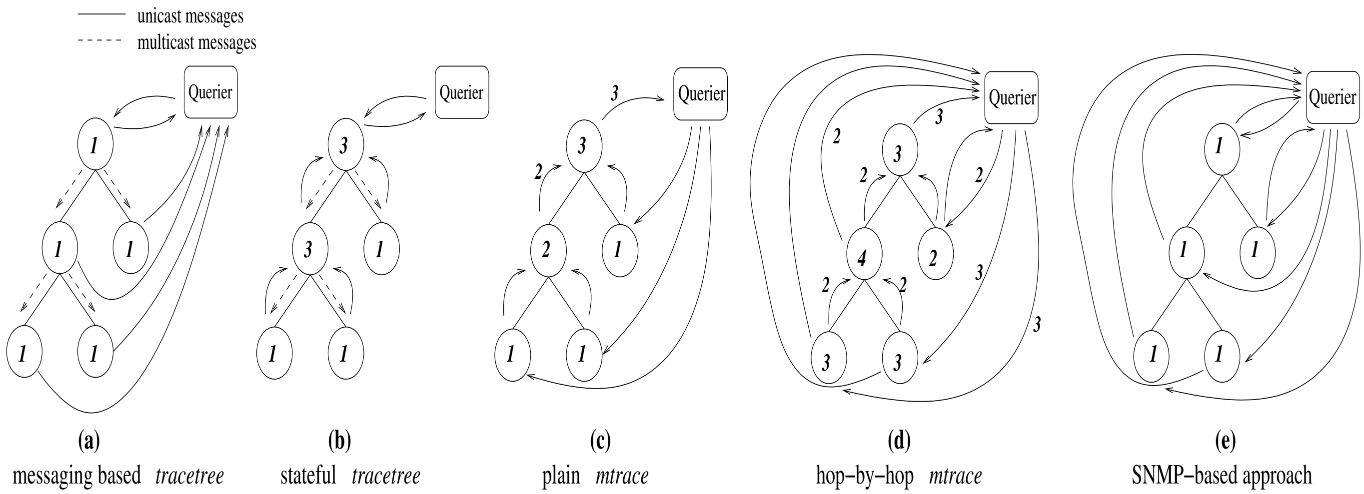

Figure 2: Comparison of topology discovery techniques.

The *mwalk* tool provides an interface that is used to generate multicast trees topologies for a given number of receivers. Using this interface we generated a number of multicast tree topologies. Since *tracetree* works with internal routers and does not deal with the session receivers, we removed the session receivers and used only the topologies for our simulations. In this paper, we provide simulation results for four topologies with sizes 141, 290, 495 and 762 nodes. These topologies were obtained by using the *mwalk* topology generator interface for 30, 100, 300 and 1000 receiver multicast trees. We have observed similar results for a number of other generated tree topologies.

5.2 Router Overhead

The processing overhead on the routers refers to the amount of work that the routers perform on receiving a topology discovery request. In the *message-based tracetree* and *mtrace* approaches, routers create a response record and send it to a given destination. In the *stateful tracetree* approach, routers perform more complex processing. They store temporary state information for requests; forward request to their downstream neighbors on the tree; and keep timers while waiting for responses from downstream links, etc. In the SNMP-based approach, routers perform a lookup operation on the related MIB tables and send responses to the management site. A more detailed comparison of the approaches in terms of router overhead would require actual analysis of prototype implementations of these new methods and is therefore beyond the scope of this paper. On the other hand, as seen in Figure 2, different techniques require different number of visits to routers. We assume that for all the approaches, the processing overhead at each visit is comparable and then look at only the number of visits to the routers. This gives us a simple yet useful metric to compare the router overhead. Figure 3 shows the total number of visits to on-tree routers using a logarithmic scale. According to this figure, the SNMP-based approach requires the least number of visits to routers. In fact, in this approach, each on-tree router is visited only once. Starting from the root router and using a network topology map, each router on the multicast path is identified and visited only once. On the other hand, *hop-by-hop mtrace* has the highest number of visits to routers. In the *hop-by-hop mtrace* approach, each multicast path is traced incrementally (similar to traceroute) and the routers on each multicast paths are visited several times. A router that belongs to more than one multicast paths is visited for each path discovery. As a result, this approach involves a significantly large number of visits to routers. In the *plain mtrace* approach, routers on a multicast path are visited only once but routers that belong to more than one multicast paths are visited during each path discovery. In the *message-based tracetree* approach, the number of router visits depends on the size of the multicast tree. As we discussed above, if the tree topology is large, the discovery process is divided into rounds. In this case, a number of routers may be visited more than once (once for every round they participate in). In the *stateful tracetree* approach, a router is visited several times. The number of visits is equal to the number of links that this router has on the multicast tree.

5.3 Messaging Overhead

In this subsection we compare the messaging overhead at the management site for various topology discovery tech-

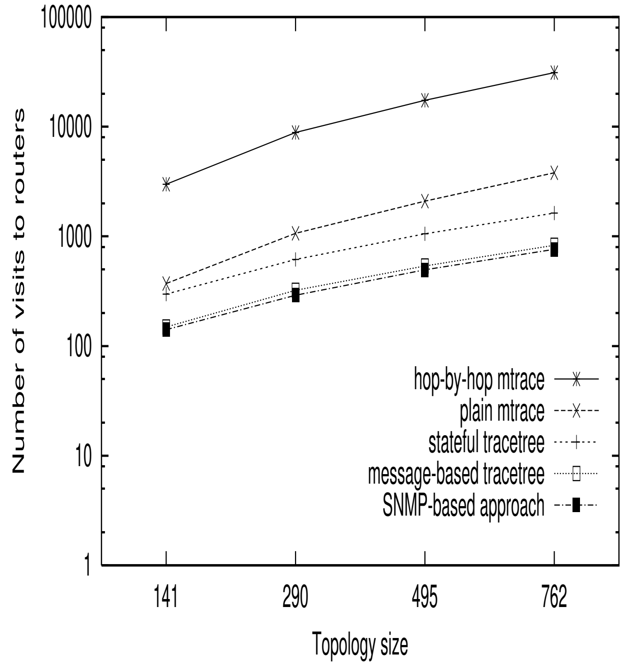

Figure 3: The number of visits to on-tree routers.

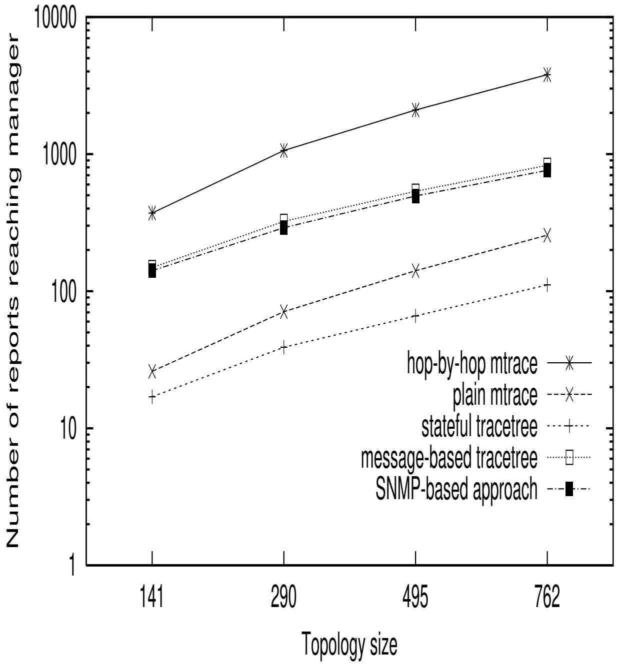

Figure 4: The number of reports reaching the management site.

niques. Figure 4 shows the number of response messages reaching the management site during the topology discovery process. According to this figure, the *stateful tracetree* technique has the least message overhead. In this approach, each response message includes a number of response records received from on-tree routers. For small-sized trees where the topology can be discovered in one round, the number of response messages reaching the management site is equal to one. As the tree size increases, the number of rounds and therefore the number of response messages reaching the management site increases. Similar to the previous case, the *hop-by-hop mtrace* approach has the highest message overhead. As we mentioned before, in this approach each multicast path between a leaf router and the root router is traced separately and the number of response messages for each path discovery is equal to the number of routers on this path. Routers belonging to several multicast paths send response messages several times. Therefore, as the tree size increases, the number of messages reaching the management site increases significantly. The *plain mtrace* approach performs the second best in terms of the message overhead. In general, in this approach the number of messages reaching the management site is equal to the number of receivers in the session. But in our simulations we do not use the receiver information. Instead, we assume that only the leaf routers have receivers adjacent to them and therefore we run mtraces only from the leaf routers. In the real world, internal routers may also have session receivers and this may cause more mtraces for the topology discovery. In addition, receiver locality may not be always known and this may cause the same multicast path to be traced for several times. Finally, for the *message-based tracetree* and SNMP-based approaches, the number of response messages is equal to the number of router visits.

5.4 Topology Discovery Duration

The last set of simulations are conducted to compare the topology discovery times of the various approaches. As we discussed above, different techniques have different processing requirements in the routers. These differences affect the topology discovery time duration. But in our simulations, we ignore these differences and compare the techniques based on the number of rounds that they take to discover the tree topology. A round involves sending a query message by the management site and then receiving all the responses from the routers. Therefore, the duration of a round is equal to the round trip time between the management site and the router whose response arrives at last the management site. According to this definition, the time duration of different rounds may be different. For simplicity purposes we also ignore this difference in our simulations. Based on these assumptions, we count the number of rounds for all five approaches. Figure 5 shows the results. Note that this figure shows only the number of rounds to collect the overall tree topology. If the rounds are performed in a sequential manner, this figure gives the relative performance of the techniques in terms of time duration. However, in reality a number of these rounds can be run simultaneously. For example, in *plain mtrace*, the management site can send mtrace queries to all the session receiver sites simultaneously. Similarly, in the SNMP-based approach, all the downstream neighbors of router can be traced simultaneously. Similar parallelization can be used in *tracetree* based approaches. This parallelization, which we call as *phases*, can help to reduce the overall time to discover tree topologies. Figure 6 shows the number of phases to discover tree topologies using the maximum possible parallelization. According to this figure, *plain mtrace* requires only one phase to discover the topology and both *hop-by-hop mtrace* and SNMP-based approaches require the highest number of phases (which is equal to the height of the tree). Note that the improvement in the topology discovery time is gained at the expense of introducing potential scalability problems. For example, in the *mtrace*- based approaches and *stateful tracetree* approach, the routers belonging to several multicast paths will be overwhelmed by simultaneous topology discovery requests and in the *message-based tracetree* and SNMP-based approaches the management site will be overloaded with the arrival of simultaneous response messages.

5.5 Discussion

According to our preliminary evaluations above, in terms of processing and messaging overhead, the *tracetree*-based approaches are at least comparable with the SNMP-based approach and the *plain mtrace* approach and are better than the *hop-by-hop mtrace* approach. In terms of topology discovery time, the *tracetree*-based approaches are the best or the second best. In addition, *tracetree* based approaches work with the existing multicast forwarding state in the routers. In this respect, they are independent of the multicast routing algorithm used to create this forwarding state. We believe that these characteristics are encouraging in terms of ease-of-deployment in the current Internet. On the other hand, *tracetree* approaches may suffer from a number of disadvantages. The existence of non-compliant routers interferes with the topology discovery operation. *Tracetree* uses an Average Branching Factor (ABF) to correct the effect of non-compliant routers. But it may not always be easy to choose an appropriate ABF factor. If the ABF factor is too small or too large, it may cause potential scalability problems. Similar to the case with non-compliant

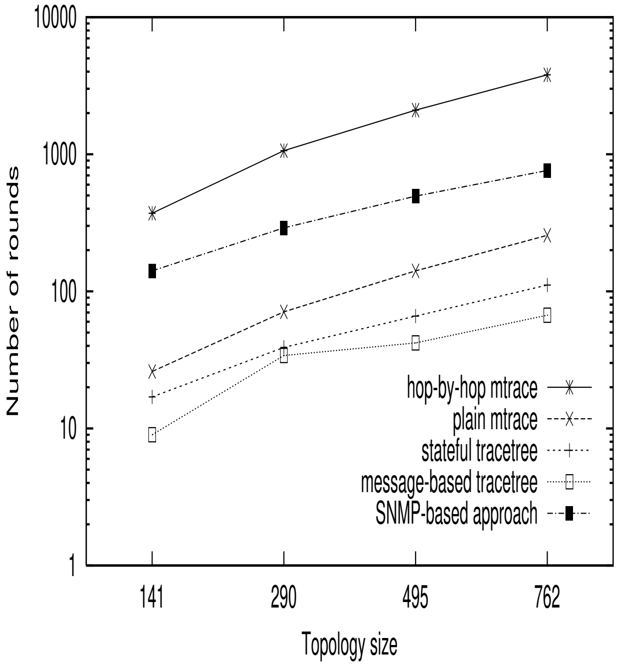

Figure 5: The number of rounds needed to discover the topology.

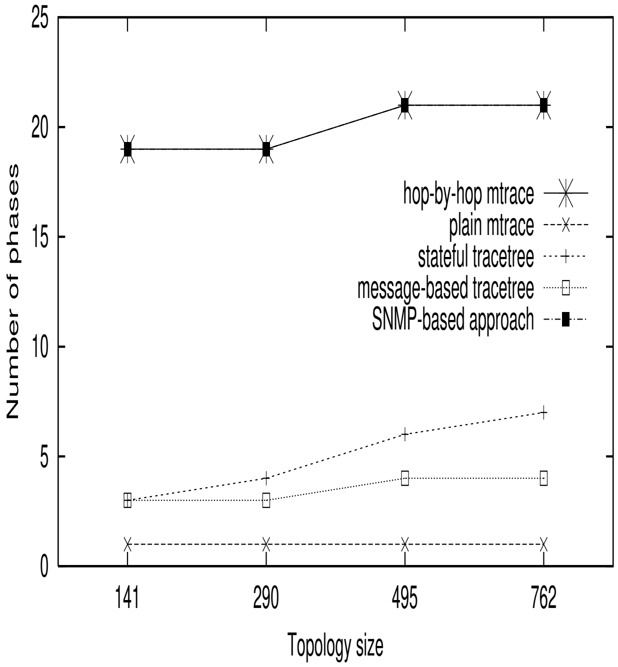

Figure 6: Number of phases to discover the topology.

routers, routers under heavy workload tend to ignore IP router alert options and forward the packets without doing any processing on them. This means that *tracetree* compliant routers under heavy workload may act as non-compliant routers. In addition, *tracetree*-based approaches return only the multicast topology information and they are insensitive to encapsulations used in some multicast routing protocols, e.g. PIM-SM. Moreover, in the *stateful tracetree* approach, routers depend on a timing mechanism in collecting reports from their downstream neighbors. However in the current Internet, packets may be delayed unexpectedly.

Another important consideration is the scalability of *tracetree*-based topology collection approaches. Currently, *tracetree* uses rounds to collect the *actual* tree topology information in a scalable manner. But for a very large multicast tree topologies this approach may not be the best. For tree topologies with thousands of routers, it may take a very long time to collect the *actual* tree topology information. In these scenarios, we may use a hierarchical approach to collect the tree topology. For example, we may first collect the topology information in the *domain* level and then collect the *virtual* subtree topologies for each domain and then collect the actual subtree topologies for each domain. We leave these issue for future work.

6. CONCLUSIONS

In this paper we have investigated the problem of multicast tree topology discovery. We started by motivating the need for topology discovery and presenting existing approaches. Then, we presented our approach, *tracetree*. Our approach starts a tree query at the first hop router and uses one of a number of techniques to control feedback implosion. Our approach requires relatively little additional router support and relies only on forwarding state. Unlike other approaches it does not require knowledge about group receivers or information about the general topology. A general requirement of *tracetree* is that it requires tight control on the number of request messages that are forwarded throughout the tree. In this respect, we have identified and discussed a number of issues related to our topology discovery approaches. In addition, we conducted a number of simulations to evaluate the performance of our techniques. We compared our techniques to *mtrace*-based and SNMP-based techniques in terms of router processing overhead, network utilization overhead and the time spend in topology discovery. We have seen that our techniques perform relatively better than the other alternatives. We believe that our techniques provide a scalable way of discovering a multicast tree's topology in realtime while requiring marginal additional functionality in routers.

7. REFERENCES

[1] K. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment," *IEEE Network*, vol. 14, pp. 10–20, January/February 2000.

[2] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, pp. 10–20, January/February 2000.

[3] 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*, March 2001.

[4] D. Makofske and K. Almeroth, "Real-time multicast tree visualization and monitoring," *Software–Practice & Experience*, vol. 30, pp. 1047–1065, July 2000.

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

[6] R. Malpani and E. Perry, *mmon: A multicast management tool using HP OpenView*, December 1999. Available from http://www.hpl.hp.com/mmon/.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications." Internet Engineering Task Force (IETF), draft-ietf-avt-rtp-new-*.txt, November 1998.

[8] H. Holbrook and B. Cain, "Source-specific multicast for IP." Internet Engineering Task Force (IETF), draft-holbrook-ssm-arch-*.txt, March 2001.

[9] D. Katz, "IP router alert option." Internet Engineering Task Force (IETF), RFC 2113, February 1997.

[10] R. Chalmers and K. Almeroth, "Modeling the branching characteristics and efficiency gains of global multicast trees," in *IEEE Infocom*, (Anchorage, Alaska, USA), April 2001.

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