

Tracetree: A Scalable Mechanism to Discover Multicast Tree Topologies in the Internet

Kamil Sarac, *Member, IEEE* and Kevin C. Almeroth, *Senior Member, IEEE*

Abstract—The successful deployment of multicast in the Internet requires the availability of good network management solutions. Discovering multicast tree topologies is an important component of this task. Network managers can use topology information to monitor and debug potential multicast forwarding problems. In addition, the collected topology has several other uses, for example, in reliable multicast transport protocols, in multicast congestion control protocols, and in discovering network characteristics. In this paper, we present a mechanism for discovering multicast tree topologies using forwarding state in the network. We call our approach *tracetree*. First, we present the basic operation of *tracetree*. Then, we explore various issues (e.g., scalability, security, etc.) related to its design and deployment of *tracetree*. Next, we provide a detailed evaluation by comparing it to the currently available alternatives. Finally, we discuss a number of deployment issues. We believe that *tracetree* provides an efficient and scalable mechanism for discovering multicast tree topologies and therefore fills an important void in the area of multicast network management.

Keywords—Multicast, tree topology discovery, forwarding state.

I. INTRODUCTION

With the deployment of native multicast in commercial networks, multicast is getting closer to become a ubiquitous service in the Internet. However, before multicast can be used as a revenue-generating service, its robust and flawless operation needs to be established in the inter-domain[1]. This requires the availability of management tools to help network administrators configure and maintain multicast functionality within and between multicast-enabled domains.

Discovering multicast tree topologies is an important component of multicast network management[2]. Network managers can use the topology information as the basis of group monitoring, or can use it to identify potential multicast forwarding problems that may occur due to routing protocol limitations, multicast network mis-configurations, or routing policy decisions. In addition, topology information has several other uses: reliable multicast transport protocols[3], multicast congestion control protocols[4], and discovering network characteristics[5]. Finally, end users can use topology information and traffic flow to monitor activity in a group, or, if there is a problem, where to direct an inquiry[6].

There are mainly two different approaches to discovering multicast tree topologies: (1) inference-based approaches and (2) network-probing approaches. An inference-based ap-

proach uses loss and/or delay as seen from receiver sites to infer the logical tree topology[7], [8]. A *logical* tree is an abbreviation of a multicast tree that includes only the root, the branching, and the leaf routers on the multicast tree. From the point-of-view of network management and debugging multicast forwarding problems, this approach is only of limited value. First, it can only return the logical tree topology rather than the actual tree topology. Second, it requires all receivers to participate in the topology inference operation.

In the second approach, topology information is collected directly from the routers in the network. This can be done in two different ways: (1) using the Management Information Base (MIB) table information, and (2) using the Routing Information Base (RIB) table information in the routers. In the first method, using the Simple Network Management Protocol (SNMP), routers can be queried to return group-specific information from their MIB tables. This information can then be used to build a representation of the tree topology. In the second method, the packets used to discover the tree topology are forwarded by the routers based on the routers' RIB table information. This method is divided into two parts based on the kind of information used: (a) routing-based and (b) forwarding-based approaches. In multicast, routing information is used to create a multicast tree between receiver sites and the source. During this process, each on-tree router creates and adds a new entry into its *multicast forwarding table*. This forwarding state contains the interface in which the multicast data is expected to arrive and the interface(s) on which the multicast data is to be forwarded. Therefore, using the RIB table information in routers, the tree topology can be discovered in two different directions:

- *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 is used to build a tree[9], [10]. This approach requires knowing the identities of all session receivers.
- *Source-to-receiver(s) direction*: Multicast *forwarding* information is used to discover the tree topology. Topology discovery starts from the root of the multicast tree and progresses toward the receivers. In this approach, topology discovery packets are forwarded by the routers on the tree based on their multicast forwarding state. Therefore, this approach does not require knowledge of session receivers.

Currently, there are mechanisms (e.g., *mtrace*[10] support in the routers) and proposed tools (e.g., *MHealth*[9]) to discover multicast tree topologies in the receiver(s)-to-source direction. However, topology discovery in the source-to-receiver(s) direction has been considered impractical due to scoping difficulties and many-to-one scalability problems.

Kamil Sarac is with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, (email: ksarac@utdallas.edu).

Kevin Almeroth is with the Department of Computer Science, University of California, Santa Barbara, CA 93106, (email: almeroth@cs.ucsb.edu).

Contrary to this belief, our main purpose in this paper is to show that topology discovery in this direction is not only viable but also more effective, more efficient, and more scalable when compared to existing alternatives. In this paper we present our topology discovery approach as a stand alone mechanism and compare it to existing approaches such as *mtrace* and SNMP-based approaches. However, the main focus of this paper is not necessarily to define a protocol specification for standardization purposes, but to present the feasibility and advantages of our approach. In addition, in order to clearly present the advantages and disadvantages of our approach, we introduce it independent of the currently existing management frameworks, namely SNMP-based network management. However, the mechanisms presented in this paper can be included in a network management platform, such as HP OpenView[11], and co-exist with SNMP. In summary, we see our work as a feasibility study and leave the development of the actual protocol(s) (either a stand alone protocol or an extension to the SNMP-based management framework) to the Internet Engineering Task Force (IETF).

In this paper, we explore the possibilities of using forwarding state to discover multicast tree topologies. We call our approach *tracetree*. Compared to the *existing mtrace* and SNMP-based techniques, *tracetree* provides a more efficient and more scalable mechanism to collect multicast tree topologies in the network. We present the basic idea behind *tracetree* and discuss a number of important issues related to its functionality. In addition, we provide a detailed evaluation of the *tracetree* mechanism. Finally we discuss a number of deployment issues for *tracetree* in the Internet.

The remainder of the paper is organized as follows. The next section is on related work. Section III presents the basic operation of *tracetree*. Section IV discusses issues related to *tracetree*-based topology discovery. Section V and VI address scalability and efficiency concerns for *tracetree*. Section VII includes our evaluation of *tracetree*. Section VIII addresses a number of deployment issues and the paper is concluded in Section IX.

II. RELATED WORK

In order to evaluate *tracetree*, it is important to understand how related techniques operate. Therefore, in this section, we study the currently available techniques for topology discovery. There are a number of existing multicast management tools that collect multicast tree topologies for monitoring purposes. These tools can be grouped into SNMP-based tools and *mtrace*-based tools. In this section, we briefly describe *mmon* and *MHealth* as example tools in these groups respectively.

Mmon is an SNMP-based software tool developed at HP Labs[12]. It was developed primarily for managers of IP multicast networks. *Mmon* uses a network map and a number of multicast related MIB tables to discover a multicast tree's topology. In practice, SNMP-based MIB information in routers are accessible only by Network Operation Center (NOC) personnel within an administrative domain. Therefore, *mmon* is well-suited for use by NOC personnel within a domain. However, it is not accessible by ordinary end users in

the domain and it is also not usable to discover multicast tree topologies that span multiple domains in the network. As a result, *mmon* (or any other SNMP-based topology discovery tool) is effective at discovering multicast (sub)tree topologies within a domain only.

MHealth, the Multicast Health Monitor, is an *mtrace*-based multicast monitoring tool[9]. It works in the receiver(s)-to-source direction and uses *mtrace* and the Realtime Transport Control Protocol (RTCP)[13]. *Mtrace*[10] is a multicast version of the *traceroute* utility. It is used to discover the multicast path between a given receiver and a source in a multicast group. The trace starts at the receiver site and works in the reverse direction toward the source site. *Mtrace* has two modes of operation: (1) *plain mtrace* and (2) *hop-by-hop mtrace*. 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 complete path information. If *plain mtrace* is not successful, hop-by-hop mode is used. *Hop-by-hop mtrace* works similarly to *traceroute*. Requests start with a TTL of one and is incremented after successfully receiving a response from the routers. One requirement of the *mtrace*-based approaches is to know the identities of each and every receiver in a multicast group. *MHealth* uses RTCP reports to collect this information. RTCP is defined as part of the Realtime Transport Protocol (RTP). RTCP specifies periodic transmission of control packets by all group members to all other group members.

An advantage of *MHealth* over the SNMP-based tools is that it is a user-level tool and it can work on an inter-domain scale. On the other hand, *MHealth* depends on RTCP data which is unreliable (i.e. one may not learn a complete set of receivers in a multicast group). In addition, emerging multicast protocols, e.g., Source Specific Multicast (SSM)[14], break the RTCP mechanism by allowing only the session source to transmit data to the group multicast address. Finally and most importantly, *MHealth* can only be used to discover tree topologies where the application layer transport mechanism uses RTP.

III. *Tracetree* FUNCTIONALITY

In this section, we give an overview of the *tracetree*-based topology discovery mechanism. Our general approach is to discover tree topologies in the source-to-receiver(s) direction. This approach depends on using the forwarding state information in the routers to discover multicast tree topology. A *querier* that is interested in discovering the tree topology sends a *tracetree query* message to the root router and then expects to receive *tracetree responses* from the on-tree routers. On receiving the query message, the root router first creates a response message and sends it to the querier via unicast. Then, the router creates a *tracetree request* packet and forwards it on the multicast tree. 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 . On receiving this query message, A changes it to a *tracetree request*

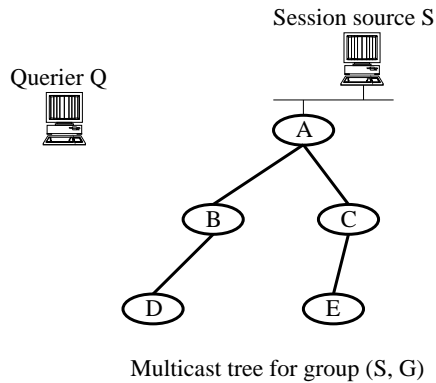


Fig. 1. Overview of topology collection.

packet and forwards it to its downstream neighbors on the multicast tree (routers B and C). *Tracetree* requests include a *tracetree protocol header*. The root router A puts its own IP address as the *request forwarder* and the IP address of the querier Q as the *response destination* into the *tracetree* protocol header.

Each on-tree router, upon receiving a *tracetree* request packet, sends its own response back to the querier. In this response, the router includes the address of the request forwarder router, the local interface address it received the request packet from, and the number and addresses of interfaces that it forwards the request packet on. If the scope of the request packet expires, it reports this fact in the response as well. In addition to sending its own response information back to the querier, the router replaces the request forwarder address in the incoming request packet with its own IP address and forwards the request packet down the tree as long as the scope of the packet allows. During this process, the querier collects the incoming response messages to create a representation of the multicast tree topology.

Finally, depending on the type of the multicast tree, the root router may be either the first hop router at the session source site (in the case of source specific trees) or it can be a Rendezvous Point (RP) router (in the case of shared trees). In addition, the querier can send its queries to any on-tree router to discover the sub-tree topology rooted at this router. Similarly, in the case of bi-directional shared trees (such as CBT), the on-tree router receiving a *tracetree* query forwards the *tracetree* request messages away from the core toward the receivers.

IV. TOPOLOGY DISCOVERY ISSUES

While the basic operation of *tracetree* is straightforward, there are a number of additional issues that need to be considered. We discuss these issues below.

A. Forwarding Request Packets

Tracetree request packets are forwarded based on existing multicast data forwarding states. The actual forwarding can be done in two different ways: (1) hop-by-hop forwarding of request packets, or (2) multicast forwarding of request packets. In the hop-by-hop forwarding approach, each on-tree router forwards the request packet to its downstream neighbors similar to *mtrace* request forwarding discussed in the previous sec-

tion. That is, if the forwarding router knows the IP addresses of downstream neighbors on the tree, it sends the *tracetree* request packet to these addresses individually. However, if it does not know the IP address of a downstream neighbor, or if the outgoing interface is on a shared media, it sends the *tracetree* request to a link-local multicast group address defined by the multicast routing protocol in use. In addition, request forwarding can use acknowledgements for reliability.

In the case of multicast forwarding approach, the request packets are addressed to the multicast group address. In order to distinguish *tracetree* requests from regular data packets, the routers set the IP Router Alert option[15] in the *tracetree* request packets. This option forces routers to pull the request packets from the forwarding fast path and examine them. In addition, in the case of source specific tree discovery, routers spoof the IP address of the session source and use it as the source IP address in the request packet. This enables request packets to successfully propagate on the source specific forwarding tree. In the case of a shared tree discovery, this spoofing may not be necessary for tree discovery but would be useful to avoid creating additional forwarding state entries in the routers.

The main advantage of the hop-by-hop forwarding is that it does not use IP Router Alert option. In general, Router Alert option processing is considered to be expensive as it causes routers to closely examine packets that are not directly destined to themselves. On the other hand, the disadvantage of this approach is that during the initial deployment of *tracetree* in the network, a router that does not implement the *tracetree* functionality would not know how to handle the request packets and therefore drop them. This would essentially halt the topology discovery at this router leaving the subtree beyond it undiscovered. In the case of multicast forwarding of request packets, non-compliant routers would simply forward the request packets without doing any processing on them. As we discuss in Section IV-C, this would cause some performance degradation in topology discovery but would enable us to continue the topology discovery beyond non-compliant routers.

A potential deployment scenario may use a hybrid approach as follows. When forwarding the request packets on a point-to-point link, a router uses the hop-by-hop forwarding approach with reliability. If the downstream neighbor does not send an acknowledgement back, then the forwarding router sends another copy of the request packet using the multicast forwarding approach. However, when forwarding request packets on a shared link, routers send two copies of the requests; one using hop-by-hop approach, and the other using the multicast forwarding approach. In this case, compliant routers will process the first and ignore the second, and non-compliant routers will discard the first (the hop-by-hop one) and forward the second (the multicast one).

B. Scalability

In *tracetree*, a given query message may result in a large number of *simultaneous* responses sent to the querier. If not controlled, these responses may cause an implosion problem at the querier. In order to control the number of responses,

we divide the topology discovery process into rounds and discover only a controlled portion of the multicast tree in each round. Therefore, within each round, we would like to control the total number of responses sent to the querier by limiting the number of routers receiving a *tracetree* request packet. One way to control the scope of IP packets is to use standard TTL-based scoping. The semantics of standard TTL-based scoping is that the given TTL value indicates the maximum number of hops that an IP packet can propagate on the multicast tree. Accordingly, the number of routers receiving a request packet may change based on the shape of the multicast tree within the given scope. For example, in the tree topology given in Figure 2-a, a TTL of 7 results in 14 responses while the same TTL value results in 32 responses for the topology in Figure 2-b. This example shows that the standard TTL-based scoping is not very helpful in accurately controlling the number of response messages. Instead, we need a scoping mechanism in which the TTL value indicates the maximum number of routers that should receive a copy of a *tracetree* request packet. In our work, we propose a modification to the standard TTL-based scoping mechanism. According to our *modified-TTL-based* scoping mechanism, the scope of request packets is controlled using the IP TTL field but routers use a different computation to decrement the value. More specifically, routers use their outgoing degree (i.e. the number of outgoing on-tree interfaces) to compute the new TTL value, TTL_{new} , as

$$TTL_{new} = \lfloor \frac{TTL_{current} - 1}{num_neighbors} \rfloor. \quad (1)$$

In this computation, $TTL_{current}$ is the TTL value of the incoming *tracetree* request packet and $num_neighbors$ is the outgoing degree of the router on the multicast tree. According to this computation, the $TTL_{current}$ value limits the number of routers that should receive a copy of the request packet. In the case of the first hop router, it may use either a pre-configured default value or a $num_response$ value that it receives from the querier as the initial value for $TTL_{current}$. In the example in Figure 2-c, the given TTL value 7 results in 7 routers receiving a request packet. In this case, the remaining portion of the tree is discovered in additional rounds. The querier sends new query messages to appropriate on-tree routers to continue the topology discovery.

C. Existence of Non-compliant Routers

In this section we discuss the effect of non-compliant routers on topology discovery. Non-compliant routers are the routers that do not support *tracetree* functionality. Since *tracetree* depends on routers to participate in topology discovery, non-compliant routers may cause scalability problems in the response collection process. That is, non-compliant routers use the standard TTL decrement mechanism when forwarding *tracetree* request packets.

In order to detect the existence of non-compliant routers, we use a technique based on the duplication of the TTL value in request packets. On forwarding a request packet, routers

copy the IP TTL value into a field, TTL_{tt} , in the *tracetree* protocol header. Upon receiving a request, routers compare the values in the IP TTL and the TTL_{tt} fields. The difference between these two values gives the number of non-compliant routers since the last compliant router on the path. *Tracetree* must now decide what should be done.

The first compliant router after the non-compliant router(s) uses an *Average Branching Factor* (ABF) to re-compute the IP TTL value, TTL_{IP} , of the incoming request packet. The ABF value is supplied by the querier in the initial query message and is carried in the *tracetree* header in request packets. The assumption here is that, on the average, each router in the network has ABF outgoing interfaces on the multicast tree. By using ABF and TTL_{tt} , routers can compute a modified TTL'_{IP} value as

$$TTL'_{IP} = \frac{TTL_{tt}}{(ABF)^{(TTL_{tt}-TTL_{IP})}}. \quad (2)$$

After updating the TTL value of the incoming request packet, routers continue with the normal *tracetree* request processing.

Regarding the computation of ABF value, the querier can use its own statistics based on the branching characteristics of previously discovered tree topologies. As a querier performs *tracetree* queries, it continuously updates its ABF statistics based on the branching characteristics of the collected tree topologies and uses this information in future *tracetree* requests. This will help the querier to dynamically update the ABF value as the underlying multicast network topology changes in the Internet. In a recent study, Chalmers and Almeroth report that the ABF for internal multicast routers is approximately 1.42[16]. A querier with no past *tracetree* experience can use this value to start. Our evaluations in Section VII-C show that ABF of 1.42 is quite effective in controlling report implosion at the querier site.

Similar to the case with non-compliant routers, the existence of multi-access links between on-tree routers may cause irregularities in scope calculations. This is mainly because routers may not know the number of on-tree routers sharing the same multi-access link with themselves. In this case, routers can use the number-of-neighbors information that they maintain as part of the multicast routing protocol to compute the TTL_{IP} value in the *tracetree* request packets.

D. Security

Security is an important concern in *tracetree*. This is because *tracetree* could be used to launch third-party denial-of-service attacks. A malicious user could spoof the IP address of a third-party site and identify itself as a querier causing routers to send a potentially large number of responses to the victim site. In order to make launching this type of attacks difficult, we use a three-way handshake for *tracetree* queries. In this mechanism, a router R applies a hash function H_K with a periodically changing secret key K on the IP address of the incoming query message to generate a string S . Then, R sends S back to the IP source of the query message. Later on, when R receives a response from the querier with the correct string

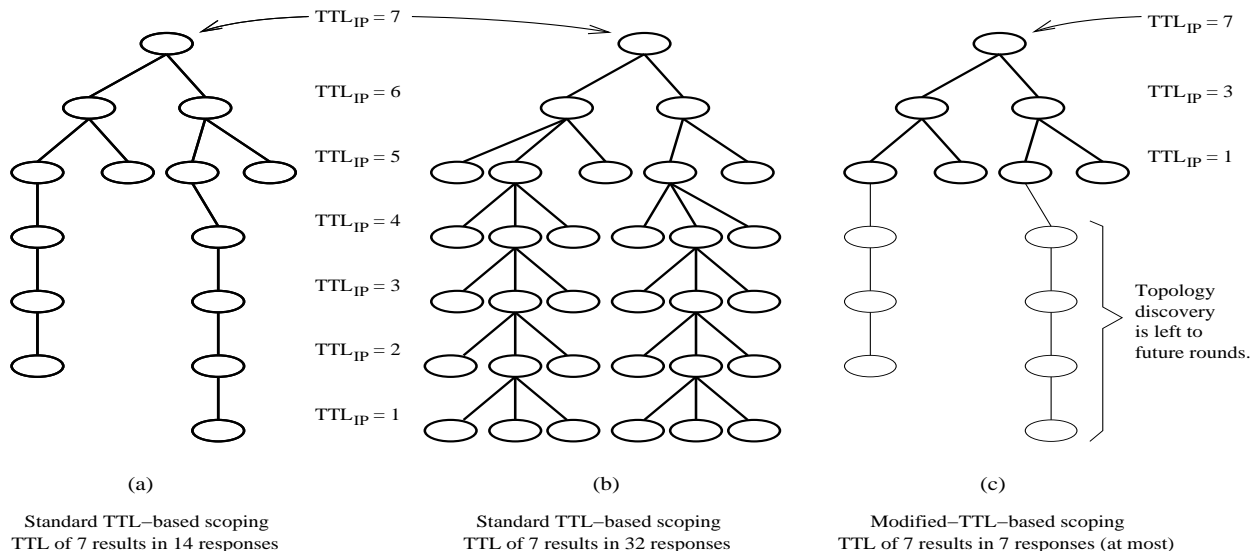


Fig. 2. Comparison of TTL-based scopings.

S , which it can verify by applying H_K^{-1} on S , it honors the request. If a router changes its key value while there are pending query messages to be verified, the router keeps the previous key (or a few of them) to authenticate a reply message coming from the querier. Therefore, when the router receives a message with a string S that it cannot verify with the current key, it uses the previous key to verify it.

A second mechanism that *tracetestree* uses for protection is to regulate the query/request processing rate at routers. That is, a router receiving a large number of queries/requests in an interval may ignore some or all of them. This mechanism both prevents the router from being overloaded with query/request messages and reduces the effect of potential attacks.

Another security issue for *tracetestree* 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) rule[17].

E. Finding the First Hop Router

Before sending a *tracetestree* 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 should be easy to determine. Remote queriers can run a group specific *mtrace* toward the session source and learn this information. Considering the possibility that there may be multiple routers on the source's LAN, after identifying a first hop router through *mtrace*, the querier may use IGMP ASK_NEIGHBORS queries to learn the addresses of other first hop routers and may send additional queries to them. Alternatively, the router could send the *tracetestree* request on the LAN, and the request will be forwarded by all other first-hop routers. However, if the session source is connected to multiple LAN segments, the above mechanism may not be that helpful in learning the addresses of first hop routers on different LAN segments.

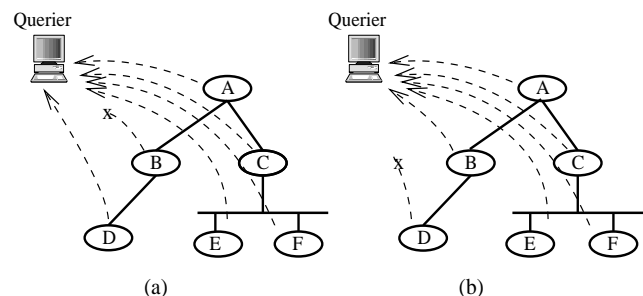


Fig. 3. Loss of response messages.

F. Loss of Request or Response Messages

The loss of a request message results in the premature end of the discovery round. This typically requires the querier to run an additional round to trace the missed tree branch. The loss of a response message may have different implications depending on the location of the loss within a round. For example, in Figure 3-a, the loss of the response message coming from router B causes a gap in the tree topology collected at the querier site. In this case, the querier can send a new *tracetestree* query to the appropriate on-tree router(s) (to router A in the above example) to retrieve the missing information. On the other hand, when the response message of a leaf router (leaf of a round or leaf of the tree) gets lost, as in Figure 3-b, this does not cause any gaps in the collected topology. When router B sends its response, it reports the fact that it forwarded the request message on its outgoing interface on the tree. Therefore, the querier expects to receive a response from a router reporting that this router received the request packet from router B. Due to the lack of such a response, the querier decides that this response was lost and therefore runs a new round to collect this information by sending a new query to router B. In general, the querier may need to send several such query messages to a number of on-tree routers to collect the missing information.

G. Effect of Topology Changes During Discovery

Multicast forwarding trees are created and maintained in the network dynamically. Changes in the underlying unicast network topology may necessarily change tree topologies in the network. These changes may in turn cause query or request messages to arrive at routers that are no longer on the multicast tree. In the case of compliant routers, they can send a message back to the querier informing it of the situation. This way, the querier explicitly learns about the topology changes in the network and re-runs *tracertree* to discover the new tree topology. In the case of non-compliant routers, they can simply ignore the message. Depending on the policy that the querier uses for handling non-compliant routers, it may take longer for the querier to learn/detect the topology changes in the network.

V. IMPROVING *Tracertree* SCALABILITY

Tracertree depends on each and every compliant on-tree router to send its response back to the querier. Basic scalability is provided by dividing topology discovery into rounds and discovering a controlled portion of the tree in each round. In addition to this mechanism, based on the characteristics of multicast forwarding trees, we propose a new response collection approach to further improve the scalability of *tracertree*. We call this approach *non-relay response collection (nr-response)*.

In a related recent study, Chalmers and Almeroth report that relay routers constitute more than 80% of the intermediate routers on a multicast tree[16]. Similar results have been reported in an earlier study by Pansiot and Grad[18]. Based on these findings, *nr-response* operates as follows: on receiving a request packet, each *relay* router first creates its response packet. Then, instead of sending this response directly to the querier, it appends it to the end of the request packet and forwards it to its downstream neighbor. Each *branching* router, on receiving a request packet, first creates its own response packet and appends it to the end of the accumulated information. At this point, the collected response information corresponds to the multicast path between this router and the previous compliant branching router on the multicast tree. In the next step, this router separates the accumulated response information from the request packet and sends it back to the querier. In the last step, it forwards a fresh request packet (a request packet having no response information appended) to its downstream neighbors. In addition, if a router has only one out-going interface but this interface is on a shared LAN segment and if this router has more than one multicast enabled neighbors on this shared LAN segment, than the router considers itself as a branching router. In the case of *leaf* routers, they will perform similar steps as the branching routers (except for the request forwarding step).

One final modification related to *nr-response* is on the scope calculation of the request packets. As we discussed previously, *tracertree* uses a modified-TTL scoping mechanism for scalability and uses the duplication of IP TTL values (in the TTL_{tt} field) to detect non-compliant routers. In the original *tracertree* mechanism, at each compliant router on the tree

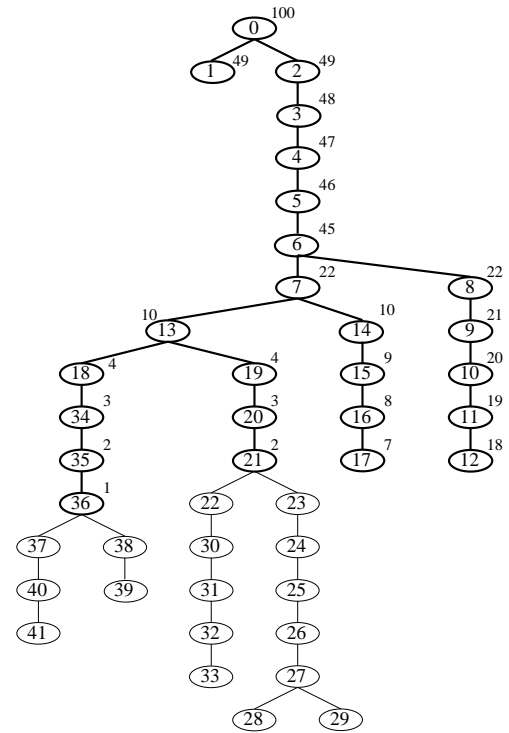


Fig. 4. A sample multicast tree topology.

(whether it is a relay router or not), these values are computed/decremented according to Equation 1. In *nr-response*, we require only the branching routers and the leaf routers to send responses back to the querier. Therefore, using the IP TTL value alone is not very helpful for controlling the number of responses. For this reason, we propose a slightly different TTL scoping mechanism for controlling the scope of request packets in the network. That is, we use a new field TTL_{nr} in the *tracertree* protocol header to indicate the number of responses expected to be received in this round from the network. In this situation, the (TTL_{IP}, TTL_{tt}) pair is used to detect non-compliant routers on the multicast tree. When a non-relay router receives the request packet, it uses the TTL_{nr} value to send its response back to the querier and modifies this value for the request packets that it forwards on the tree. In addition, in order to prevent pre-mature scope expiration (due to IP TTL expiration in the network) each compliant router on the multicast tree adjusts TTL_{IP} and TTL_{tt} values according to TTL_{nr} value.

As an example, consider the tree topology in Figure 4. According to *nr-response*, the querier will receive responses only from the root router (0), branching routers (nodes 6, 7, 13, 21, 27, and 36) and leaf routers (nodes 1, 12, 17, 28, 29, 33, 39, and 41). Therefore, the querier will learn the exact same topology information but will receive fewer responses (15 responses instead of 42 in this particular example). Thus, based on branching characteristics, we can reduce both the number of rounds and the overall discovery time.

VI. EFFICIENCY CONSIDERATIONS

So far, we have presented a new approach for tree topology discovery in the source-to-receiver(s) direction and discussed a number of important issues that are critical to its ef-

fective operation. One other issue that we need to address is the efficiency of our mechanism. We define three efficiency metrics for *tracetestree*: (1) time to trace a tree, (2) number of rounds to trace the tree, and (3) the number of responses returned in a round for a given initial TTL value. In this context we are interested in finding tight bounds for these metrics. These bounds will then give us a means to evaluate the efficiency of *tracetestree*. In general, we would like to discover a given tree topology in the shortest possible time. This time usually depends on the number of rounds that it takes to discover the topology. In addition, the number of rounds depends on the number of responses collected in each round. Finally, the number of responses collected in each round depends on the initial TTL value. Overall, all of these metrics depend on the size, depth, and branching characteristics of the given tree topology.

Tracetestree provides a spectrum of choices for response collection. In general, during topology discovery, on-tree routers may not know the branching and depth characteristics of the subtree below them. This makes it difficult for an arbitrary response collection mechanism (i.e. for an arbitrary TTL scoping procedure) to bound the number of responses.

We now present an example response collection mechanism that gives reasonably good bounds on the above mentioned efficiency metrics. According to this mechanism, the querier starts the topology discovery with an initial TTL value of one. In this round, the querier learns the number of children that the queried router has on the tree. In the next round, the querier uses a TTL value large enough to reach the children of this router on the tree. At the end of this round, the querier receives responses from the routers on the second level (children of the first queried router). At this point, the querier sends parallel query messages to each of these routers with TTL values large enough to reach their children only. We call these parallel rounds *phases*. Therefore, by running parallel queries, the querier discovers the tree topology level by level until it reaches edge routers at the receiver sites. Consequently, in this mechanism, the number of responses in each round (efficiency metric 3 above) is bounded and this bound is equal to the given TTL value.

In this topology discovery approach, the number of rounds is equal to the depth of the tree, $depth_{max}$. But this information may not be known at the querier site initially. Instead, we can approximate this value by the maximum path length in the network, $depth_{network}$. This gives us an upper bound on the number of rounds (efficiency metric 2 above) needed for topology discovery. Finally, in terms of the topology discovery time, due to the parallelism used, this time is bounded by $depth_{max} * RTT_{max}$ where RTT_{max} is the maximum round trip time between two systems in the network. Since the querier does not know $depth_{max}$ for the tree, we can approximate it with $depth_{network}$. As a result, $depth_{network} * RTT_{max}$ gives an upper bound for the tree topology discovery time (efficiency metric 1 above).

This response collection mechanism works very similar to the SNMP-based topology discovery mechanism and therefore it may have similar efficiency behavior. However, as we

will see in Section VII, when used with *nr-response*, this approach out-performs all of the alternative approaches in all of the efficiency metrics.

VII. EVALUATIONS

In this section, we evaluate *tracetestree*. First, we study the effect of several parameters on topology discovery. These include multicast tree shape, the initial TTL value of *tracetestree* request packets, and the existence of non-compliant routers in the multicast tree. In addition, we compare *tracetestree* to other alternatives in terms of topology discovery time and topology discovery overhead. Most of our evaluations are based on simulations. At the end of the section, we provide a discussion justifying the reasons for our evaluation approach. In the evaluations, when appropriate, we simulate the topology discovery techniques using the ns-2 network simulator[19]. We generate realistic multicast tree topologies using two different data sets collected from the Internet. We start the section by explaining these data sets.

A. Data Set

In our simulations, we use two sets of multicast tree topologies. The first set of trees are generated using *mwalk*[16] with realistic multicast network topology maps. These maps were collected by tracing multicast paths between three different source sites and a large number of receiver IP addresses[16]. The sources were located at Georgia Tech (GaTech), UC-Santa Barbara (UCSB), and University of Oregon (UofO). The receiver IP addresses have been known to have participated in multicast groups at some point over the last 10 years[20]. The *mwalk* tool provides an interface that is used to generate multicast tree topologies for a given number of receivers. The tool randomly chooses a number of IP addresses as receivers and then constructs a multicast tree. Since *tracetestree* works with internal routers and does not deal with receivers, we remove the session receivers and use only the resulting tree topologies for our simulations. Using *mwalk*, we were able to generate multicast trees with sizes up to 900 nodes.

The second data set was collected by running unicast traceroute queries from our site at UT Dallas to a large number of remote sites. We used the collected unicast path information to create multicast tree topologies. Our preliminary analysis on the degree, depth and branching characteristics of these tree topologies showed close similarity to the previously reported multicast tree characteristics[16]. The main motivation for generating the second data set has been to generate realistic multicast tree topologies with larger sizes. For this, we used network prefix information from BGP routing table entries and tried to randomly generate valid IP addresses from these network prefixes. Then, for each IP address, we checked if the address is valid by using the unicast *ping* tool. Next, we tried to run traceroute queries to collect the network path toward each address. We collected around 2,500 alive IP addresses and only 1,222 of them had complete traceroute information (i.e. all the routers on the path returned ICMP TIME_EXCEEDED responses). We used the collected uni-

cast path information to form a *mother* tree and generated a number of multicast trees from this tree similar to the *mwalk* approach described above. Ideally, using this approach, we could have collected large tree topologies. However, due to high memory requirements, we were not able to run simulations with tree topologies larger than 1,554 nodes.

For each tree topology, the querier is located at the root of the multicast tree. In addition, we use BGP tables to map IP addresses to their corresponding AS numbers. Based on this mapping, we divide the tree topologies into a backbone network and edge networks and assign random link delays from 0-5 ms interval to edge network links and random link delays from 5-20 ms interval to backbone network links. Figure 4 shows an example tree topology collected by *mwalk*.

B. Effect of Tree Shape on Topology Discovery

In this subsection, we look at the effect of tree shape on the performance of *tracetest* topology discovery. In *tracetest*, a request packet is assigned an initial TTL value. This value indicates the maximum number of responses to be retrieved from the routers in one round. However, due to the shape of the multicast trees, the actual number of responses received may be smaller. This is related to scope calculations performed at on-tree routers. Since on-tree routers do not know the topology of the multicast sub-tree below themselves, they optimistically divide the remaining number of expected responses equally among their downstream neighbors. However, a branch of the sub-tree rooted at a router may have fewer routers than the expected number of responses from that branch. This situation contributes to the difference between the expected and the actual number of responses collected in a round. For example, in Figure 4, we have a sample tree topology with 42 routers. The number adjacent to each link shows the IP TTL values of the *tracetest* request packets when they arrive at these routers. For our topology discovery procedure, these numbers also indicate the number of responses expected from the sub-tree below these links. According to this figure, we see that the *tracetest* request packets have TTL values larger than one when reaching the leaf routers 1, 12, and 17. The topology discovery prematurely stops due to scope expiration at routers 21 and 36. Even though the initial TTL value (100) is larger than the size of the multicast tree (42), the shape of the tree and the lack of knowledge at individual routers prevents us from discovering the tree topology in one round.

One approach to reduce the difference between the initial TTL value and the number of returned responses is to use parallel queries with smaller TTL values. The intuition here is that as we decrease the TTL value, the possibility that *tracetest* request packets with large TTL values reach leaf routers decreases. Therefore, the number of returned responses gets closer to the initial TTL value. First, the querier sends the very first query with the initial TTL value and gets a number of responses from on-tree routers. Then, the querier identifies a number of on-tree routers to send additional query messages. At this point, instead of sending new queries sequentially to each of these on-tree routers with the initial TTL value, the querier divides the initial TTL value among the new queries

and sends the queries to the set of routers simultaneously. This way, the querier runs a number of parallel queries and still controls the amount of feedback coming from on-tree routers. Note that this approach essentially reduces the number of sequential *tracetest* queries and consequently the time required to discover the tree topology. This is desirable in cases where we need to run periodic *tracetest* queries to detect changes in a multicast tree topology.

In order to observe the effect of initial TTL value on topology discovery, we run simulations with different initial TTL values (TTL values of 25, 50, 75, 100, 150, 200, 255). Figure 5 shows the ratio of the number of returned responses to a given initial TTL value (*response-to-TTL ratio*) for TTL values of 25, 100, and 255 for both data sets. According to this figure, as the initial TTL value decreases, the number of returned responses approaches the initial value. Considering the fact that multicast forwarding trees can be significantly unbalanced, these results support our arguments on using parallel *tracetest* queries with smaller TTL values. The difference observed in two figures is due to the specific characteristics of the two data sets. *Mwalk* data depends on the current/recent multicast backbone topology in the Internet. Compared to the overall Internet topology, the existing multicast backbone is smaller and the number of multicast nodes with close proximity to each other is greater. Therefore, most of the individual tree topologies generated with this data set had few receivers that were very close to the root (e.g., see node 1 in Figure 4). As we mentioned above, due to the TTL assignment for *tracetest* request packets, such receivers affect the response-to-TTL ratio. On the other hand, in the traceroute data set, the IP addresses were chosen from a much larger set and the number of receivers close to the source was much smaller. This observation suggests that the response-to-TTL ratio is likely to improve as the multicast becomes a ubiquitous service in the Internet.

C. Effect of Non-Compliant Routers on Discovery

Tracetest non-compliant routers may affect topology discovery. As we discussed before, non-compliant routers use the standard TTL decrement operation and this may interfere with the scoping mechanism used for *tracetest* request packets. In this part of our evaluation, we run simulations to observe the effect of non-compliant routers on the number of responses collected in a round. In the simulations, we select a group of routers as non-compliant with probabilities 10%, 50%, and 75%. For this we use two different approaches. In the first approach, we randomly choose a number of routers as being non-compliant. This approach simulates a case where *tracetest*-enabled routers are being incrementally deployed by all ISPs. In the second approach, we first randomly choose a number of Autonomous Systems (ASes) that the multicast tree spans and then mark all the routers in these ASes as non-compliant. This simulates an inter-AS level partial deployment.

In our simulations, we run *tracetest* starting from the root of the tree with an initial TTL value of 100. Initially, we ran simulations to get the number of responses in a normal operation case (i.e. 0% non-compliant case). In the second step, we

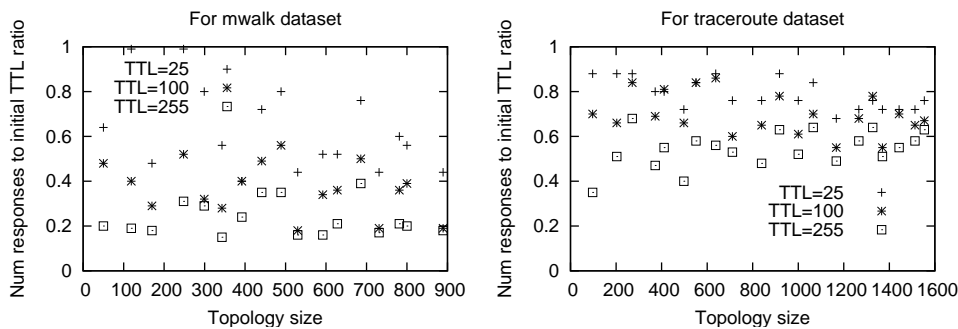


Fig. 5. Num. of responses to initial TTL ratio.

ran simulations to see the effect of different ABF values on controlling feedback implosion. We used three ABF values: 1, 1.42, and 2. These values correspond to under estimating, correctly estimating (1.42 is the average ABF value reported in [16]), and over estimating the ABF values respectively. According to Figure 6, when the non-compliant rate is small (10%), the ABF-based TTL re-computation seems to be quite effective. In addition, when the ABF value is under or over estimated, the results do not seem to cause significant variations in the number of responses returned to the querier. If we assume that this case (10% non-compliant rate) corresponds to a wide scale deployment of *tracetree*, the results clearly show the effectiveness of ABF-based TTL re-computation. On the other hand, as the rate of non-compliant routers increase, the selected ABF value plays more significant role in controlling the response implosion. An under estimation (ABF=1) usually results in significantly large number of responses, especially for the larger tree topologies. On the other hand, an over estimation (ABF=2) usually causes significant reductions in the number of responses, especially for smaller tree topologies. Finally, according to the figures, for most of the tree topologies the ABF of 1.42 results in smaller number of responses and therefore helps with preventing response implosion. We also note that non-compliant routers do not always result in an increase in the number of responses.

D. Implementation of Topology Discovery Approaches

In this subsection, we describe our implementations of the alternative topology discovery approaches. We evaluated an improved version of *mtrace*, the standard SNMP-based approach, and the *nr-responding* version of *tracetree*.

As a general policy, we try to discover a tree topology in the shortest possible time. For this we use the maximum possible parallelization for each alternative approach. However, in order to prevent response implosion, we divide the topology discovery task into rounds and use a *maximum threshold value* for the expected number of responses in each round. A round includes sending the query messages to the network and collecting all the responses. In the case of *mtrace* and SNMP, the maximum threshold value limits the number of responses received from on-tree routers. In the case of *tracetree* approach, it limits the initial TTL values used in the request packets. Having mentioned this policy, next we briefly describe the implementation of each alternative approach.

***Mtrace*-based approach:** For each round, the querier sends parallel *hop-by-hop mtrace* queries to the edge routers at the receiver sites. The number of such queries is limited by the maximum threshold value. That is, at each round, we send out at most *threshold* many queries and expect to receive at most that many responses. In this approach, a path discovery stops at a branching router on the multicast tree that has already been traced by another *mtrace* request. This prevents us from incurring redundant overhead during the topology discovery.

SNMP-based approach: In this approach, starting from the first hop router at the session source site, the querier sends SNMP queries to on-tree routers and learns the identities of down stream routers. In the next round, it sends parallel queries to these routers to learn similar information. In this way, the topology discovery progresses level by level. The number of parallel queries is limited by the maximum threshold value.

***Tracetree-nr* approach:** This approach uses the *nr-response* mechanism in which the relay routers append their responses to the request packet and forward it on the tree. Non-relay routers send (partially) collected tree topology information back to the querier. First, the querier sends a query to the first hop router with a TTL_{nr} value that is equal to *maximum threshold value*. The first hop router sends its response back to the querier and then forwards a *tracetree* request packet on the tree. Relay routers append their information to the request packets and forward them on the tree. Branching and leaf routers send the accumulated path information back to the querier. The information each branching/edge router sends essentially includes path information from the root router (or the previous branching router) to the current router. In addition, each branching router includes its out-going degree. After receiving these responses, the querier sends new query messages to these branching routers. Based on the branching information that the querier learns from the incoming responses, it divides the *maximum threshold value* among the branching routers and sends parallel queries to these routers. Therefore, the number of parallel queries in this approach is controlled such that the number of expected responses is limited by the *maximum threshold value*.

E. Topology Discovery Time

In this section, we compare the topology discovery times of the alternate approaches. Topology discovery time highly

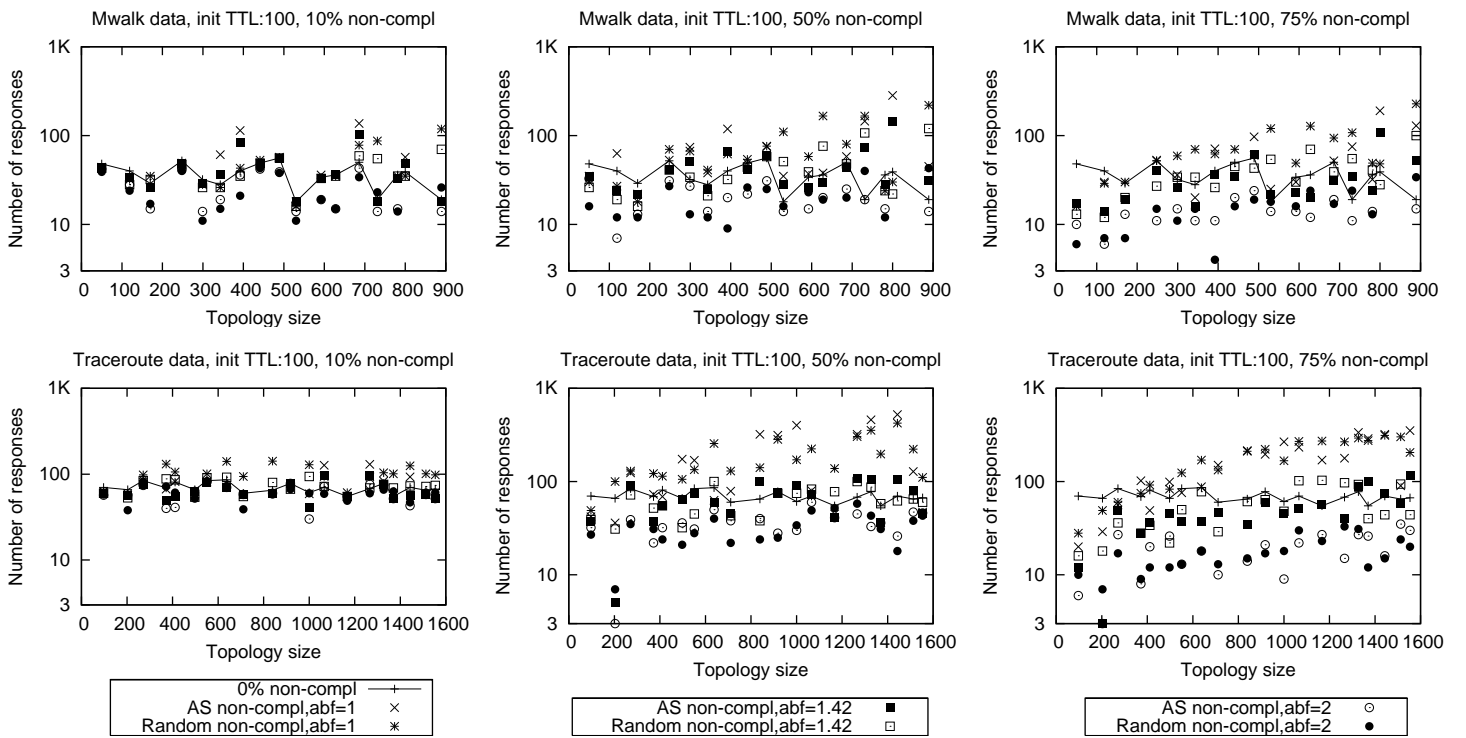


Fig. 6. Effect of non-compliant routers.

depends on the shape of the multicast tree and a number of parameters specific to each approach. More specifically, in the case of *tracetree*, the branching factor of internal tree routers, the number of responses requested in each round (by the querier), and the round parallelization factor affect the topology discovery time. Similarly, for the *mtrace*-based approach the number of leaf routers, and for the SNMP-based approach the branching factor of internal tree routers affect the topology discovery time.

For comparison, we run simulations on our sample tree topologies. In the simulations, we use the *maximum threshold values* as a common parameter across all the alternative approaches. We run simulations with six different threshold values: 25, 50, 75, 100, 200, and 255. It is expected that as the threshold value gets larger, the topology discovery time shortens. In general, we would like to discover tree topologies in the shortest time while preventing response implosion at the querier site. From this perspective, having threshold values as a simulation parameter provides a means to perform a fair comparison among the alternative approaches. But, in reality, achieving the maximum parallelism bounded by the threshold value may not always be possible. In the case of *mtrace* and SNMP, the number of nodes that can be queried/traced in parallel may be less than the given threshold value. In the case of *tracetree-nr*, we may encounter a different situation. According to Figure 5, for large initial TTL values (100 or 255), the number-of-responses-to-TTL ratio (*response-to-TTL ratio*) may be relatively small (e.g., it is less than 0.6 for the *mwalk* data set). Based on this observation, for cases in which we want *tracetree-nr* to return a relatively large number of responses (100 or more), we can choose the TTL value by considering the *response-to-TTL ratio* and increase the TTL

value by some small factor. Even though we do not use this scaling in the simulations, we believe that the performance of *tracetree-nr* could be further improved.

In Figure 7 we show the results of the simulations for the cases with *maximum threshold values* of 25, 100, and 255 only¹. Other cases present similar behavior. According to the figures, the *mtrace*-based approach takes the longest time for a majority of the cases and it is followed by the SNMP-based approach. Even though increasing the *threshold* value reduces the topology discovery time for these approaches, it does not seem to affect the relative performance between them for the majority of the tree topologies. In addition, the increase in the *threshold* value does not affect the topology discovery time for relatively small tree topologies. This is because the *threshold* value of 25 or 100 is already sufficient to achieve the maximum potential parallelism for these cases. Finally, we remind the reader that using the *mtrace*-based approach requires that we know the identities of all the session receivers in the group. However, in practice, this is likely to be very difficult (see Section II).

The *tracetree-nr* approach performs better than the other two approaches. In addition, its performance improves as the *threshold* value increases. The performance gain in *tracetree-nr* is due to the fact that only the non-relay (branching and leaf) routers send their responses back to the querier. According to our preliminary analysis, non-relay routers form, on average, 18% of the topology size in our sample tree topologies. This is also in accordance with the finding of the two previously mentioned studies [16], [18].

¹Due to large memory requirements, we were not able to successfully complete SNMP-based simulations for large tree topologies (trees with more than 1,280 nodes) for the traceroute data set.

One factor that may affect the above results is the possibility of request and/or response message losses during topology discovery. These losses may cause delays in topology discovery. In general, due to the parallelism that we use in each of the alternative approaches, it is difficult to evaluate the effect of such losses for an arbitrary case. However, in the worst case, loss of a request or a response message delays the topology discovery time by one round for all approaches.

F. Topology Discovery Overhead

In this section, we compare the tree topology discovery techniques based on their overhead. We divide the topology discovery overhead into three parts: (1) router overhead, (2) querier overhead, and (3) network overhead. The router overhead refers to the amount of work that routers perform upon receiving a topology discovery request.

In order to understand the relative performance of the various approaches, we examine the number of visits to on-tree routers. We assume that for all approaches, the processing overhead at each visit is comparable. Then, we compare the total number of visits to on-tree routers. This gives us a simple yet useful metric to compare router overhead. In the SNMP-based approach, each router is visited only once. In the case of *tracetree*, each router is visited once during a round. If the discovery process takes more than one round, each additional round increases the number of visits by one. In the case of the *mtrace*-based approach, routers on a multicast path are visited one or more times depending on their location in the multicast tree.

Figure 8-a shows the total number of visits to routers for our sample data sets for an initial TTL of 100. Other cases (TTL of 25 and 255) presents similar results. According to the figures, the *mtrace*-based approach has the largest number of visits to the routers in all of the cases. For the SNMP-based approach, the number of visits is equal to the topology size. *Tracetree-nr* performs close to the SNMP-based approach.

Querier overhead refers to the number of responses received by the querier during topology discovery. In the SNMP-based approach, this is equal to the topology size. In *tracetree-nr*, it is given by the sum of the number of branching and leaf routers and the number of rounds to discover the tree topology. Finally, for the *mtrace*-based approach, the total is equal to the number of (*mtrace*) rounds to discover the topology. Figure 8-b shows this overhead for our sample tree topologies used in the simulations.

Finally, network overhead refers to the number of messages, including query and response messages, exchanged between the querier site and on-tree routers during topology discovery. Figure 8-c presents the network overhead of the various approaches. According to the figures, *tracetree-nr* incurs significantly less overhead than the others.

G. Discussion

Our evaluation so far has consisted mainly of simulations. We feel that our work would also benefit from analytical evaluations in understanding various protocol characteristics.

However, in our evaluation we found that *tracetree* operation strongly depends on various characteristics of the multicast tree topologies including the branching characteristics of the routers, the depth characteristics of receivers, and the number and the location of non-compliant routers on the tree. Since these characteristics vary across different multicast tree topologies, it is difficult to build a statistical model of *tracetree* operation. In this section we use some assumptions to develop a closed form formula to compute the number of rounds to discover a tree topology based on a given initial TTL value TTL_{init} .

For our analysis, we assume that multicast trees are balanced trees and internal tree nodes have an out-degree of ABF. We also assume that we have a very good estimate about the depth of the tree, *tree_depth*. Even though these assumptions are unrealistic, they enable us to derive basic analytical results about *tracetree* performance. For a given TTL_{init} and ABF values, assuming a balanced tree, the equation

$$TTL_{init} = \sum_{i=0}^{n_1} ABF^i \quad (3)$$

holds. From this equation, we can compute the level of the tree n_1 that *tracetree* request can reach as

$$n_1 = \log_{ABF} \left(\frac{TTL_{init}(ABF - 1) + 1}{ABF} \right). \quad (4)$$

After collecting the responses in the first round, the querier will continue the topology discovery by sending additional query messages to routers at level n_1 . The number of routers, m , at level n_1 is given by

$$m = ABF^{n_1}. \quad (5)$$

Based on this, the querier will compute the TTL values for the queries that it sends to the routers at this level as

$$TTL_1 = \frac{TTL_{init}}{ABF^{n_1}}. \quad (6)$$

In general, the number of levels that *tracetree* spans in a round is given by

$$n_i = \log_{ABF} \left(\frac{TTL_i(ABF - 1) + 1}{ABF} \right). \quad (7)$$

Considering the fact that

$$TTL_i = \frac{TTL_{init}}{ABF^{\sum_{j=0}^i n_j}}, \quad (8)$$

we have

$$n_i = \log_{ABF} \left(\frac{TTL_{init}(ABF - 1) + 1}{ABF^{1 + \sum_{j=0}^i n_j}} \right). \quad (9)$$

Finally, when

$$\sum_{i=1}^k n_i = tree_depth, \quad (10)$$

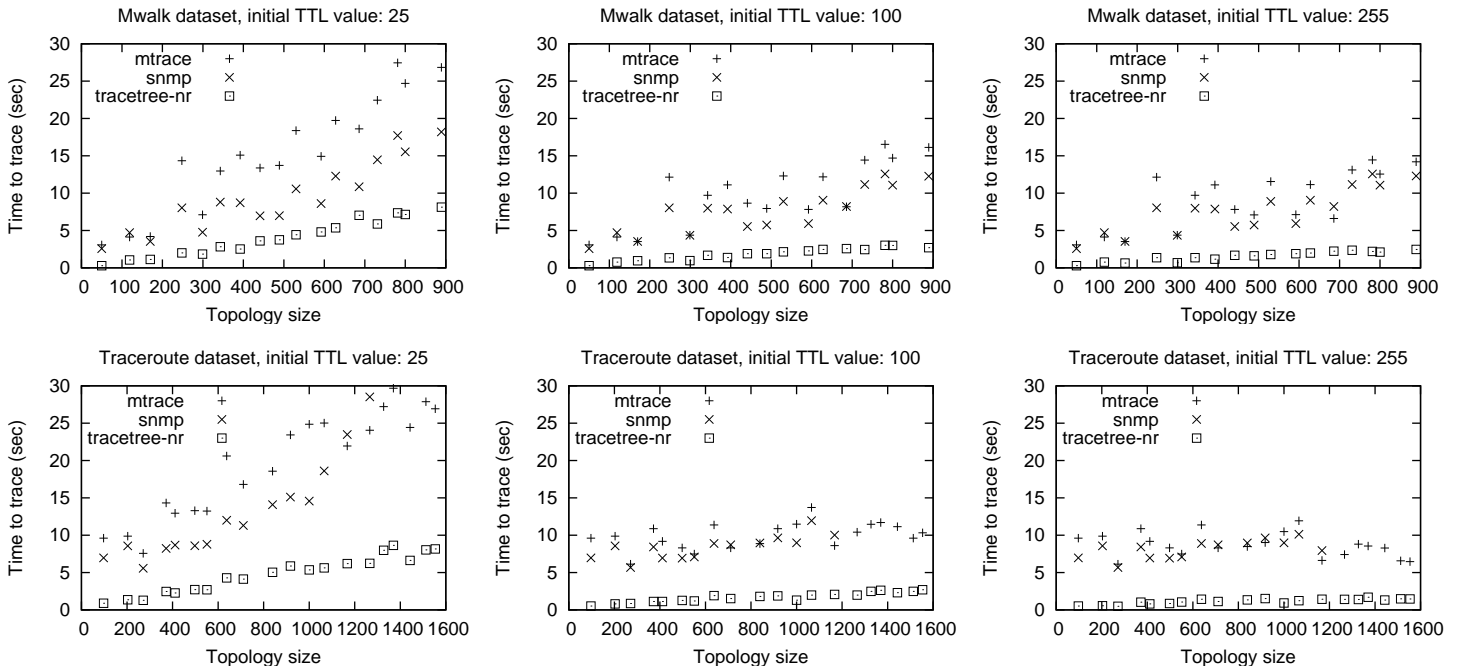


Fig. 7. Time to discover tree topologies (in seconds).

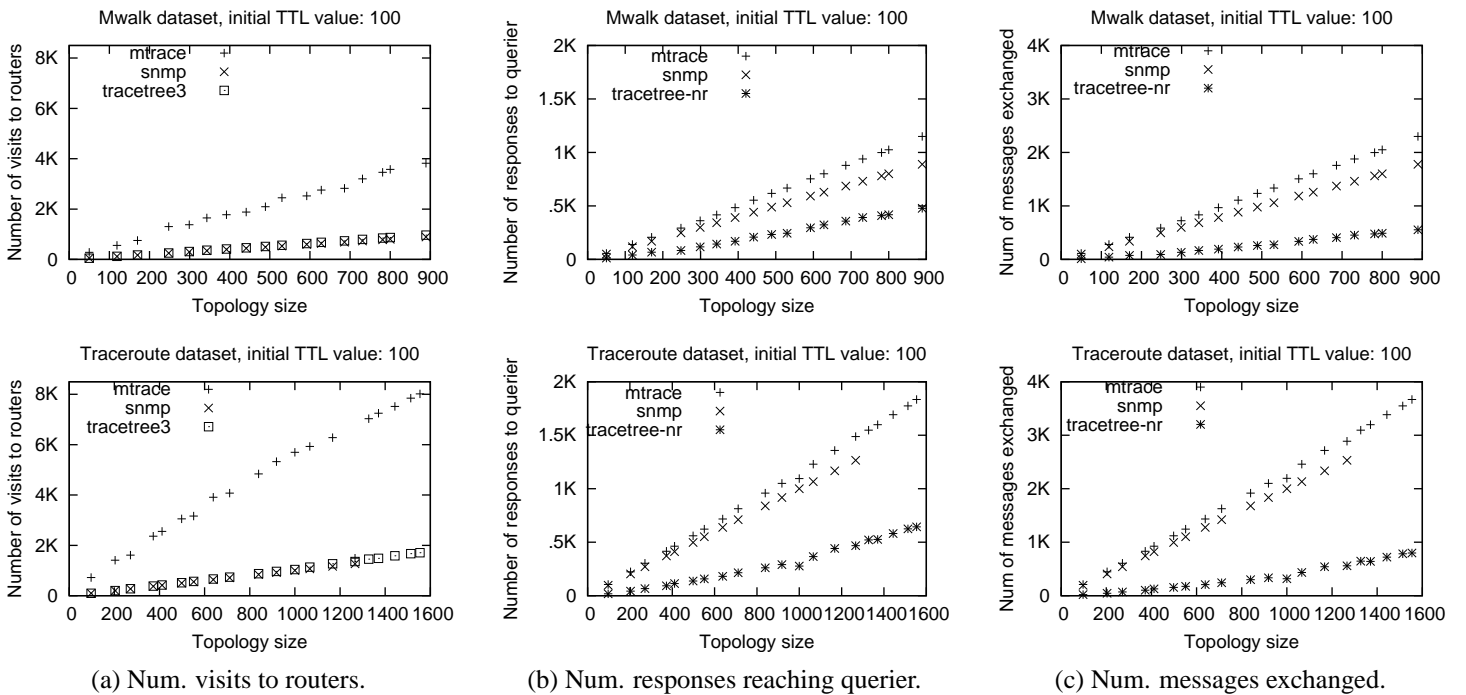


Fig. 8. Overhead comparison.

k gives the number of rounds to discover the topology.

According to the above derivation, by using Equations 9 and 10, a querier can estimate the number of rounds and therefore the time to discovery a tree's topology. But one difficulty in this approach is that the above equality depends on certain assumptions we made at the beginning of our derivation. These assumptions may not always hold for the actual tree topologies used in the multicast applications.

Finally, an experimental evaluation of *tracetree* requires a large scale deployment to see its performance under various conditions presented in the above evaluations. A prototype implementation would only show the basic proof-of-concept

but would not go far enough to reveal the true challenges of deploying *tracetree*. Because of these difficulties, our evaluations have mainly focused on conducting simulations on realistic multicast tree topologies.

VIII. DEPLOYMENT ISSUES

In this section, we discuss potential *tracetree* deployment issues. One important issue is security in terms of using *tracetree* for launching denial-of-service attacks. This is possible if the *tracetree* functionality is accessible by any user. In Section IV-D, we presented mechanisms to make launching attacks more difficult and discussed how to reduce the effect of

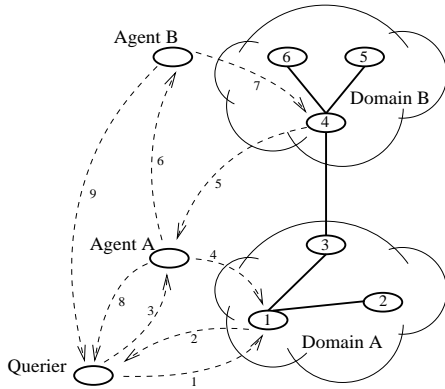


Fig. 9. Agent-based *tracetree* operation.

1. Querier sends a query packet to first hop router 1.
2. The first hop router 1 responds with the address of the *tracetree* agent Agent A.
3. Querier sends its query to Agent A.
4. Agent A sends a query to first hop router 1. During topology discovery, on-tree routers in Domain A send their responses back to Agent A.
5. When router 4 receives a request from router 3 it sends the address of Agent B to request destination i.e. Agent A.
6. Agent A forwards query to Agent B.
7. Agent B sends a query to router 4. During topology discovery, on-tree routers in Domain B send their responses back to Agent B.
8. Agent A sends the collected responses back to the querier.
9. Agent B sends the collected responses back to the querier.

Fig. 10. Topology discovery steps for Figure 9.

potential attacks. Ideally we expect these measures to provide sufficient assurance for the deployment of *tracetree* in the Internet. However, these mechanisms may not always be satisfactory for all users (ISPs). Considering this possibility, instead of completely turning off *tracetree* functionality in routers, concerned users can use a more controlled operation environment for *tracetree*. In this scenario, we use an agent-based *tracetree* topology collection mechanism similar to the Multicast Consolidated Proxy Monitor (MCPM)[6]. Figure 9 shows the steps and Figure 10 describes each step. In this approach, each domain allocates a well-known *tracetree* agent responsible for running all *tracetree* queries in the local domain. All the routers are configured to accept *tracetree* query messages only from the local *tracetree* agent in their domain. Since *tracetree* is limited to supporting requests coming from a well-known agent site, secure communication primitives can be used to provide authenticated message exchange between the agent site and the routers. Once a *tracetree* agent receives a query packet, it runs the query in the local domain, collects the responses, and sends them back to the querier. In cases where a tree topology spans multiple domains, *tracetree* agents in adjacent domains communicate query messages between each other so that a *tracetree* agent in each domain traces the portion of the multicast tree in its own domain and then sends a response back to the original querier. In this case, from

the querier's point-of-view, a given query results in responses equal to the number of domains spanned. We believe that this number should be small and therefore should not pose a substantial threat in terms of denial-of-service attacks. In addition, in an agent-based deployment scenario, *tracetree* agents can cache the collected topology and use this information for subsequent queries. Moreover, agents can perform additional operations such as hiding the actual IP addresses of the routers in order to protect privacy of the internal network topology. In summary, even though we prefer a native/standard deployment for *tracetree*, we expect the agent-based deployment to provide a reasonably good assurance for ISPs to support this service in their networks.

Another important issue is the interaction of *tracetree* with the multicast routing protocols. *Tracetree* uses existing multicast forwarding states in the routers. The multicast routing protocol deployed in the network may be using unidirectional or bidirectional trees and may be building source specific or shared trees. In either case, *tracetree* discovers a (sub)tree rooted at a queried on-tree router with respect to a given multicast source in the group. In this respect, for multicast routing protocols that use the RPF forwarding rules (such as DVMRP and PIM), *tracetree* is independent of the multicast routing protocol used to create the forwarding states in the routers. For the specific case of Core Based Trees (CBT) [21] protocol, due to lack of source-specific incoming interface information at on-tree routers, *tracetree* discovers the overall forwarding tree rooted at the queried on-tree router.

Tracetree is insensitive to packet encapsulations used in some of the multicast routing protocols such as the Multicast Source Discovery Protocol (MSDP)[22] and Protocol Independent Multicast-Sparse Mode (PIM-SM)[23]. In MSDP, when a new source starts sending to a multicast group, the Rendezvous Point (RP) in the source domain uses MSDP Source Announcement (SA) messages to announce this new source to the RPs in remote domains. Later on, when the group receivers in these remote domains learn the existence of this new source, they use PIM-SM to establish a multicast forwarding path toward this new source. Therefore, *tracetree* cannot effectively return the actual multicast tree topology between this new source and the remote group receivers until the underlying forwarding tree is established. Similarly, in the case of PIM-SM, a first hop router at a new source site encapsulates the packets and tunnels them to the RP. If and when a *tracetree* request is forwarded in this way, it cannot discover the path between the first hop router and the RP directly. However, the querier can easily collect this information by running an mtrace between the two.

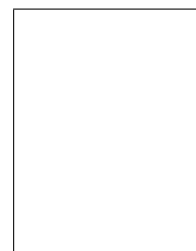
Finally, *tracetree* depends on routers to participate in topology discovery. From this perspective, during the initial deployment of the service we may have a large number of non-compliant routers in the network. In this situation, *tracetree* may not be very effective. However, as with every other new service that is developed and deployed in the network, it is reasonable to expect that after a transition period, *tracetree* will become a default function provided by all router vendors, and, therefore when deployed in a network, it will perform well.

IX. CONCLUSIONS

In this paper, we have proposed a mechanism, *tracetree*, for multicast tree topology discovery. *Tracetree* requires relatively little additional router support and relies only on forwarding state. We argued that the alternative approaches (SNMP and *mtrace*-based approaches) have requirements or limitations that significantly limit their use for topology discovery. A benefit of *tracetree* is that it provides tight control on the number of request messages that are forwarded throughout the tree. In this respect, we discussed a number of issues related to *tracetree*-based topology discovery. In addition, we have evaluated *tracetree* by comparing it to the alternative approaches. We have shown that *tracetree* is comparable or superior to the alternative approaches in terms of topology discovery overhead and topology discovery time. In addition, *tracetree* can be used in both intra- and inter-domain and it can tolerate the existence of non-compliant routers in the multicast tree. We believe that our technique provides a scalable and efficient way to discover a multicast tree's topology in real-time while requiring marginal additional functionality in routers.

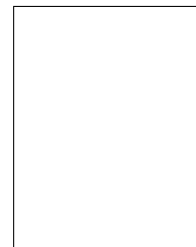
REFERENCES

- [1] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, no. 1, pp. 10–20, January/February 2000.
- [2] 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.
- [3] S. Paul, K.K. Sabnani, J.C. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 407–421, April 1997.
- [4] S. Jagannathan, K. Almeroth, and A. Acharya, "Topology sensitive congestion control for real-time multicast," in *Workshop on Network and Operating System Support for Digital Audio and Video (NOSS-DAV)*, Chapel Hill, North Carolina, USA, June 2000.
- [5] A. Adams, R. Bu, R. Caceres, N. Duffield, T. Friedman, J. Horowitz, F. Lo Presti, S. Moon, V. Paxson, and D. Towsley, "The use of end-to-end multicast measurements for characterizing internal network behavior," *IEEE Communications*, May 2000.
- [6] A. Kanwar, K. Almeroth, S. Bhattacharyya, and M. Davy, "Enabling end-user network monitoring via the multicast consolidated proxy monitor," in *SPiE ITCOM Conference on Scalability and Traffic Control in IP Networks*, Denver, Colorado, USA, August 2001.
- [7] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *IEEE Infocom*, New York, New York, USA, March 1999.
- [8] N.G. Duffield, J. Horowitz, and F. Lo Presti, "Adaptive multicast topology inference," in *IEEE Infocom*, Anchorage, Alaska, USA, April 2001.
- [9] D. Makofske and K. Almeroth, "Real-time multicast tree visualization and monitoring," *Software—Practice & Experience*, vol. 30, no. 9, pp. 1047–1065, July 2000.
- [10] W. Fenner and S. Casner, "A 'traceroute' facility for IP multicast," Internet Engineering Task Force (IETF), draft-ietf-idmr-traceroute-ipm-*.txt, July 2000, Work in progress.
- [11] *HP OpenView Network Management Solution*, Available from <http://www.hpl.hp.com/>.
- [12] P. Sharma, E. Perry, and R. Malpani, "Ip multicast operational network management: design, challenges, and experiences," *IEEE Network*, vol. 17, no. 2, Mar-Apr 2003.
- [13] H. Schulzrinne, S. Casner, R. Frederick, and Jacobson V., "RTP: A transport protocol for real-time applications," Internet Engineering Task Force (IETF), RFC 1889, January 1996.
- [14] H. Holbrook and B. Cain, "Source-specific multicast for IP," Internet Engineering Task Force (IETF), draft-ietf-ssm-arch-*.txt, November 2002.
- [15] D. Katz, "IP router alert option," Internet Engineering Task Force (IETF), RFC 2113, February 1997.
- [16] R. Chalmers and K. Almeroth, "On the topology of multicast trees," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 153–165, February 2003.
- [17] Y. Dalal and R. Metcalfe, "Reverse path forwarding of broadcast packets," *Communications of the ACM*, vol. 21, no. 12, pp. 1040–1048, 1978.
- [18] J.J. Pansiot and D. Grad, "On routes and multicast trees in the Internet," *ACM Computer Communication Review*, vol. 28, no. 1, January 1998.
- [19] K. Fall and K. Varadhan, *ns Notes and Documentation*, UC Berkeley, LBL, USC/ISI, and Xerox PARC, Available at <http://www.isi.edu/nsnam/ns>.
- [20] K. Almeroth, "A long-term analysis of growth and usage patterns in the Multicast Backbone (MBone)," in *IEEE Infocom*, Tel Aviv, ISRAEL, March 2000.
- [21] A. Ballardie, "Core based trees (CBT version 2) multicast routing," Internet Engineering Task Force (IETF), RFC 2189, September 1997.
- [22] D. Meyer and B. Fenner, "Multicast source discovery protocol (MSDP)," Internet Engineering Task Force (IETF), draft-ietf-mboned-msdp-*.txt, November 2002, work in progress.
- [23] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, G. Liu, and L. Wei, "PIM architecture for wide-area multicast routing," *IEEE/ACM Transactions on Networking*, pp. 153–162, Apr 1996.



Kamil Sarac is currently an assistant professor in the Department of Computer Science at the University of Texas at Dallas. He obtained his M.S. and Ph.D. degrees in Computer Science from the University of California Santa Barbara in 1997 and 2002 respectively. He received his B.S. in Computer Engineering from Middle East Technical University, Turkey, in 1994. His main research interests focus on multimedia networking;

IP multicast and multicast routing protocols; inter-domain network monitoring and management and distributed systems. Particularly, he has been working on developing tools and techniques for monitoring data transfer operations between multicast-enabled networks. He is a member of both the ACM and IEEE.



Kevin C. Almeroth is currently an associate professor at the University of California in Santa Barbara where his main research interests include computer networks and protocols, multicast communication, large-scale multimedia systems, and performance evaluation. At UCSB, Dr. Almeroth is a founding member of the Media Arts and Technology Program (MATP), Associate Director of the Center for Information Technology and Society (CITS), and on the Executive Committee for the University of California Digital Media Innovation (DiMI) program. In the research community, Dr. Almeroth is on the Editorial Board of IEEE Network, has co-chaired NGC 2000 and Global Internet 2001; has served as tutorial chair for several conferences, and has been on the program committee of numerous conferences. Dr. Almeroth is serving as the chair of the Internet2 Working Group on Multicast, and is a member of the IETF Multicast Directorate (MADDOGS). He is also serving on the advisory boards of several startups including Occam Networks, NCast, and the Santa Barbara Technology Group. He has been a member of both the ACM and IEEE since 1993.