Scalable and Efficient End-to-End Network Topology Inference

Xing Jin, *Student Member, IEEE Computer Society*, Wanqing Tu, *Member, IEEE Computer Society*, and S.-H. Gary Chan, *Senior Member, IEEE Computer Society*

Abstract—To construct an efficient overlay network, the information of underlay is important. We consider using end-to-end measurement tools such as traceroute to infer the underlay topology among a group of hosts. Previously, Max-Delta has been proposed to infer a highly accurate topology with a low number of traceroutes. However, Max-Delta relies on a central server to collect traceroute results and to select paths for hosts to traceroute. It is not scalable to large groups. In this paper, we investigate a distributed inference scheme to support scalable inference. In our scheme, each host joins an overlay tree before conducting traceroute. A host then independently selects paths for tracerouting and exchanges traceroute results with others through the overlay tree. As a result, each host can maintain a partially discovered topology. We have studied the key issue in the scheme, that is, how a low-diameter overlay tree can be constructed. Furthermore, we propose several techniques to reduce the measurement cost for topology inference. They include 1) integrating the Doubletree algorithm into our scheme to reduce measurement redundancy, 2) setting up a lookup table for routers to reduce traceroute size, and 3) conducting topology abstraction and reducing the computational frequency to reduce the computational overhead. As compared to the naive Max-Delta, our scheme is fully distributed and scalable. The computational loads for target selection are distributed to all the hosts instead of a single server. In addition, each host only communicates with a few other hosts. The consumption of edge bandwidth at a host is hence limited. We have done simulations on Internet-like topologies and conducted measurements on PlanetLab. The results show that the constructed tree has a low diameter and can support quick data exchange between hosts. Furthermore, the proposed improvements can efficiently reduce measurement redundancy, bandwidth consumption, and computational overhead.

Index Terms—Topology inference, topology discovery, end-to-end measurement, measurement cost.

1 Introduction

In the recent years, overlay networks have been increasingly used to deploy network services. Examples include overlay path routing, application-layer multicast (ALM), peer-to-peer streaming and file sharing, and so on [1], [2], [3], [4], [5]. In order to build an efficient overlay network, the knowledge of the underlay topology is important. For example, two seemingly disjoint overlay paths may share common underlay links; therefore, the selection of overlay paths, without the knowledge of underlay, may lead to serious link congestion. However, it is not trivial to obtain an underlay topology through end-to-end measurements. Border Gateway Protocol (BGP) routing tables can construct an AS-level topology [6], [7]. However, the tables are stored at specific gateway routers and are not publicly available. Techniques like network tomography infer a topology based on the correlation between path properties [8], [9]. Although it is an end-to-end approach, the resulting topology is often inaccurate and unstable. Therefore, the most commonly used tool is traceroute, which can explicitly extract the router-level path between a pair of hosts [10]. In fact, it has been shown that ALM based on router-level topology information can achieve substantially low end-to-end delay, low physical-link stress, and high tree bandwidth [11], [12], [13].

Given a group of N hosts, conducting full $O(N^2)$ traceroutes among them can certainly construct an accurate topology (if we do not consider measurement noise such as anonymous routers or router alias). However, we note that a host cannot conduct lots of traceroutes at the same time. This is because a router receiving too many traceroute-probing packets may stop responding to the packets or simply discard them, which incurs measurement noise of anonymous routers. As paths starting from the same host often overlap, if a host conducts lots of traceroutes at the same time, the measurement results may contain many anonymous routers. On the other hand, traceroute may take as long as minutes to identify a router-level path. Therefore, sequential or nearly sequential traceroutes at a host result in significantly long measurement time. Furthermore, traceroute may generate many network packets. The all-pair traceroutes hence incur significant traffic in the network. In summary, all-pair traceroutes between hosts are costly and not scalable. We hence require each host to select a few destination hosts for tracerouting. With an intelligent selection mechanism, it is possible to infer a highly accurate topology with a low number of traceroutes. Note that most overlay applications do not require a 100 percent accurate topology. For example, it has been shown that the Fast Application-layer Tree (FAT) [13] achieves good performance when the topology consists

- *X. Jin and S.-H.G. Chan are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. E-mail: {csvenus, gchan}@cse.ust.hk.*
- *W. Tu is with the Department of Computer Science, University College Cork, Cork, Ireland. E-mail: wanqing.tu@cs.ucc.ie.*

Manuscript received 24 Apr. 2007; revised 24 July 2007; accepted 9 Aug. 2007; published online 6 Sept. 2007.
Recommended for acceptance by Y. Pan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-04-0124.
Digital Object Identifier no. 10.1109/TPDS.2007.70771.

of 93 percent of links [14]. A more accurate topology does not lead to a significant improvement in the tree's performance.

Previously, Max-Delta has been proposed to efficiently infer the underlay topology among a group of hosts [13], [15]. It divides the inference process into multiple iterations. In each iteration, a central server collects the traceroute results from the hosts. The server then selects some representative paths for the hosts to traceroute in the next iteration and aims at revealing as much undiscovered information on the underlay as possible. The simulation results on Transit-Stub topologies have shown that in a group of 256 hosts, each host only needs to traceroute around 10 to 14 other hosts to discover 95 percent of underlay links. This reduces the number of traceroutes by 62 percent as compared to a random measurement method and by 89 percent as compared to the full measurement method [15].

A limitation of Max-Delta is that the central server may not be able to support a large group. First, in each iteration, the server takes $O(V_p \log V_p + E_p + N)$ time to select a traceroute target for a host. Here N is the number of hosts in the group. V_p and E_p are the numbers of nodes (including the hosts and routers) and links in the already-discovered underlay topology, respectively [13]. In total, the server takes $O(N(V_p \log V_p + E_p + N))$ time to select traceroute targets for all the N hosts in one iteration. This complexity is considerably high when the group size is large. Second, the server needs to periodically accept traceroute results from all the hosts and send out traceroute targets to them. When the group size is large, the server may not be able to simultaneously set up so many connections.

In this paper, we propose a distributed inference scheme based on Max-Delta. In our scheme, each host joins an overlay tree before topology inference. A host then conducts traceroutes and distributes the results to all the other hosts through the overlay tree. As a result, each host can receive traceroute results from other hosts and maintain a partially discovered topology. Based on that, a host can use Max-Delta to select traceroute targets by itself and continue conducting traceroutes. We have studied an important issue in the scheme, that is, how the overlay tree can be constructed so that hosts can quickly exchange traceroute results. Since every host sends out traceroute results to others, we need to build a source-unspecific low-diameter overlay tree under certain degree bounds. Previous research has shown that this problem is NP-hard, and there are no efficient distributed algorithms to address it [16], [17]. We propose a distributed tree construction algorithm based on the outdegree bounds of hosts. We first identify a host as the tree root and then insert new incoming hosts around the root. A host with a larger outdegree bound is put closer to the root. A low-diameter tree is hence formed.

Furthermore, we propose several techniques to reduce the measurement cost for topology inference: 1) we note that there is high measurement redundancy among traceroute results. A router or a link may be discovered multiple times in different traceroutes. We integrate the Doubletree algorithm [18], [19], [20] into our scheme to reduce the redundancy. 2) We compress traceroute results to reduce bandwidth consumption for data exchange. Since routers are often repeatedly discovered in multiple traceroutes, we set up a lookup table to map each router (that is, router IP and router name) into an integer. Later on, we represent routers by integers. 3) As mentioned, the computational complexity of Max-Delta is considerably high when the group size is large. We reduce it by conducting topology abstraction and reducing the computational frequency.

As compared to the naive Max-Delta, the distributed scheme eliminates the central server from the system. In the scheme, the computational loads for target selection are distributed to all the hosts instead of a single server. A host only communicates with a few other hosts, and the consumption of its edge bandwidth is limited. Therefore, the scheme is fully distributed and scalable. We have done simulations on Internet-like topologies and conducted measurements on PlanetLab. The results show that the constructed tree has a low diameter, and the proposed improvements can efficiently reduce the measurement redundancy, bandwidth consumption, and computational overhead.

Note that the Internet is an asymmetric network in terms of both connectivity and distance. In a connectivity-asymmetric network, the traceroute path from a host A to another host B may not be the reverse of the path from B to A. The selection of traceroute targets in Max-Delta, and in our distributed inference scheme, is independent of connectivity asymmetry. In a connectivity-asymmetric network, we only need to regard the path from A to B and the path from B to A as two different paths. However, for ease of illustration, in this paper, we assume that the network is connectivity symmetric. Following this assumption, if the path from A to B has been traceRouted, we will not traceroute the path from B to A. On the other hand, we have used tools for estimating host coordinates in topology inference. These tools implicitly assume that the network is distance symmetric. The assumption on distance symmetry has little impact on our inference efficiency. This is because as shown in [15, Fig. 7], the inaccuracy in coordinate estimation has limited impact on Max-Delta performance.

The rest of this paper is organized as follows: In Section 2, we briefly review related work. In Section 3, we discuss the design of the distributed inference scheme. In Section 4, we discuss the improvement techniques. In Section 5, we present illustrative numerical results based on Internet-like topologies and PlanetLab measurements. We finally conclude in Section 6.

2 Background and Related Work

2.1 Previous Work on Network Topology Inference

There are many ways of inferring a network topology. Network tomography techniques periodically send probing traffic and exploit the performance in correlation to infer network topologies [8], [9]. However, because the network properties measured (for example, loss rate or delay) are often unstable and inaccurate, it is difficult to infer an accurate topology. BGP routing tables can provide AS-level information, but they usually are not available to normal hosts in the Internet [6], [7]. We hence adopt traceroute, which can obtain explicit router-level information by end hosts [10].

Traceroute is implemented with Internet Control Message Protocol (ICMP) messages. Each time, the source sends out an

IP datagram with a certain Time-to-Live (TTL) value to the destination. A router that receives the datagram is required to decrement the TTL by one before forwarding the packet. When a router receives an IP datagram whose TTL is 1, it throws away the datagram and returns an ICMP "time exceeded" error message to the source. The returned message contains the router's name, IP address, and round-trip time (RTT) from the source. In another case, if the datagram arrives at the destination with an unused port number (usually larger than 30,000), the destination generates an ICMP "port unreachable" error message and returns it to the source. Therefore, in traceroute, the source sends out a series of IP datagrams with increasing TTL to the destination, and each datagram can identify one router in the path. The whole router-level path is hence identified. All the datagrams of traceroute are sent by UDP. An outgoing datagram in traceroute contains 12 bytes of user data, 8 bytes of UDP header, and 20 bytes of IP header, for a total of 40 bytes. The size of the returned datagrams changes. The returned ICMP message contains 20 bytes of IP header, 8 bytes of ICMP header, 20 bytes of IP header of the datagram that caused the error, and 8 bytes of UDP header, for a total of 56 bytes. For each TTL value, three ICMP messages are sent. According to these data, we can compute the network bandwidth consumed by a traceroute.

Traceroute-like tools have been widely used in Internet measurements such as Skitter, Mercator, and Rocketfuel [21], [22], [23]. Skitter sends traceroute packets from different locations worldwide to actively measure the Internet topology. Mercator utilizes a modified version of traceroute to reduce the probing time. Rocketfuel combines information from BGP tables, traceroutes, and Domain Name System (DNS) to infer Internet service provider (ISP) topologies. All these work focuses on Internet-level or ISP-level topology inference, and the major concern is how we can discover a *complete* network topology, including all the routers and links. However, in our study, we are only interested in the topology among a certain group of hosts that are arbitrarily distributed in the Internet. Furthermore, we only need an approximate topology, because most overlay applications are tolerant to a small distortion of the underlay topology. For example, as shown in [14], it is enough to discover around 93 percent of links for the FAT scheme to build a tree with low delay and low bandwidth consumption. Therefore, the key problem is how the measurement cost can be reduced while keeping high topology accuracy.

Barford et al. study the relationship between the measurement accuracy and the number of traceroute sources in topology inference [24]. Given a list of destinations, they find that the marginal utility of adding additional traceroute sources declines rapidly after the second or third one. In other words, the first two or three sources can traceroute the destinations and discover the majority of the complete topology. Here, the complete topology is formed by the traceroute results from all the sources to all the destinations. In their experiments, the number of sources is much less than the number of destinations, for example, 8 sources and 1,277 destinations or 12 sources and 313,709 destinations. However, in our study, each host is a source and may be another's destination. The number of sources is equal to the number of destinations. Therefore, their conclusion may not work well in our system. We show an example by simulations on Transit-Stub topologies. We randomly put $N = 500$ hosts in the network. The complete topology is formed by all-pair traceroutes between the hosts. When the numbers of sources are 1, 2, 10, 50, 100, and 200 (for each source, the destination hosts are all the other hosts in the network), the portions of underlay links discovered are 65 percent, 70 percent, 76 percent, 84 percent, 89 percent, and 94 percent, respectively. Clearly, if we select not more than 10 sources, as suggested in [24], we only discover not more than 76 percent of underlay links. This level of accuracy is far from the aforementioned 93 percent. When we increase the number of sources to 200, we can discover around 94 percent of links. In this case, the total number of traceroutes is $\sum_{i=1}^{200}(N-i) = 79{,}900$ (assuming that paths are symmetric). On the average, each host has to traceroute 159.8 paths. As a comparison, given $N = 512$, Max-Delta only requires each host to traceroute around 40.2 paths to discover 95 percent of links (see [15, Fig. 8b]). Clearly, Max-Delta is much more efficient in discovering underlay links. Furthermore, in order to reduce the measurement time, we require hosts to conduct traceroutes in parallel. We do not use two or three hosts as sources to traceroute all the others, even though this can discover the majority of the underlay, because this takes long measurement time and puts uneven measurement loads on hosts.

A problem in traceroute measurement is that traceroute results often contain anonymous routers, which significantly distort and inflate the inferred topology. Broido and Claffy report that nearly 1/3 of the probed paths contain anonymous, private, or invalid routers [25]. They propose two methods (that is, the so-called *arcs* and *placeholders*) to bypass anonymous routers in the inferred topology. Despite their simplicity, these two methods hide much topology information and cannot efficiently reduce router inflation in the resulting topology. Yao et al. study how a topology can be inferred, given the traceroute results, where the inferred topology should be consistent with the traceroute results and contain the minimum number of anonymous routers [26]. They show that producing either an exact or an approximate solution for this problem is NP-hard. They further propose a heuristic to merge anonymous routers while keeping the consistency constraint. However, as shown in [15], the consistency check has impractically high complexity. Two fast algorithms without the consistency constraint are then proposed to merge anonymous routers and construct an approximate topology [15]. All these work on anonymous routers is orthogonal to our study in this paper. For simplicity, we have assumed that traceroute results do not contain anonymous routers. We can integrate any of the above work into our study to infer an approximate topology in the presence of anonymous routers.

2.2 Review on Max-Delta

Max-Delta considers how we can quickly and efficiently infer the router-level topology among a group of N hosts [13], [15]. As the router-level network is a sparse graph [27], it is anticipated that fully $O(N^2)$ traceroutes among hosts are not necessary. Fig. 1 shows an example, where A, B, C,

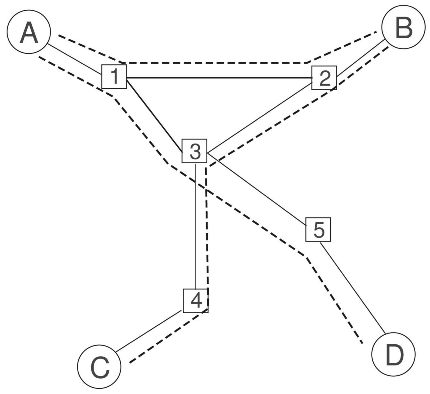


Fig. 1. An example of an underlay network. The dashed lines indicate overlay paths between hosts.

and D are hosts, and 1, 2, 3, 4, and 5 are routers. The dashed lines indicate the overlay paths between hosts. As the figure shows, paths $A - B$, $A - D$, and $B - C$ have revealed all the underlay links and, therefore, we need to measure only three, instead of all the possible six, paths.

Max-Delta works as follows: 1) Hosts utilize lightweight tools such as GNP [28] or Vivaldi [29] to estimate their network coordinates and report them to a central server and 2) the following inference procedure is divided into multiple iterations. In each iteration, the server selects a target for each host to traceroute. Hosts then traceroute their targets and report the results to the server. The server then combines all the results obtained in the iteration, and based on that, it starts the next iteration on target assignment. Such a process is repeated until a predefined stop rule (for example, a certain number of iterations) is achieved.

A target is selected as follows: For a certain host A, suppose that the path between A and another host B has not been measured. The server computes the distance between A and B in the discovered topology $D_p(A,B)$ by using shortest path routing. The server also computes the distance between them based on their coordinates $euclidean(A,B)$. Given the coordinates of A and B in a d-dimensional space, say, $(X_{A1}, X_{A2}, \ldots, X_{Ad})$ and $(X_{B1}, X_{B2}, \ldots, X_{Bd})$, we can compute $euclidean(A,B)$ as

$$\sqrt{(X_{A1} - X_{B1})^2 + (X_{A2} - X_{B2})^2 + \ldots + (X_{Ad} - X_{Bd})^2}.$$

If the coordinate estimation is accurate, $euclidean(A,B)$ will approximate the real network distance between A and B. Define the gap between these two values as

$$\Delta(A,B) = D_p(A,B) - euclidean(A,B).$$

If $\Delta(A,B)$ is large, it is with high probability that some links between A and B (leading to a shorter path in the discovered topology) are not discovered yet. For all unmeasured paths between A and other hosts, the server selects the path with the maximum Δ value as A's traceroute target.

As discussed, the scalability of Max-Delta is limited by the edge bandwidth and the computational power of the central server. Therefore, we need to design a distributed inference scheme to support large groups. We further reduce the measurement cost of the scheme from several aspects. In [30] and [31], we have proposed a distributed inference scheme based on Max-Delta and discussed how the measurement cost can be reduced. In this paper, we combine our previous work to present a complete inference scheme with high scalability and efficiency.

2.3 Review on Doubletree

Donnet et al. note that large-scale traceroute measurements such as Skitter have high intra-monitor and inter-monitor redundancies [18], [19], [20]. That is, routers and links are often repeatedly discovered in multiple traceroutes. We take Fig. 1 as an example to show the following types of measurement redundancies:

1. *Intramonitor redundancy.* Such a redundancy exists in traceroutes from one monitor (that is, the traceroute source) to multiple destinations. Suppose that A and B are two destinations and C is a monitor. When C conducts traceroutes to A and B, C obtains two traceroute paths: $C - 4 - 3 - 1 - A$ and $C - 4 - 3 - 2 - B$. Clearly, routers 4 and 3 have been visited twice in the two traceroutes. Intramonitor redundancy hence occurs.
2. *Intermonitor redundancy.* Such a redundancy exists in traceroutes from multiple monitors to the same destination. Suppose that C and D are two monitors. When they conduct traceroutes to the same destination A, they obtain paths $C - 4 - 3 - 1 - A$ and $D - 5 - 3 - 1 - A$, respectively. In this case, routers 3 and 1 have been visited twice in the two traceroutes. Intermonitor redundancy hence occurs.

Donnet et al. have proposed a Doubletree algorithm to reduce the redundancies. As discussed, a normal traceroute sends a series of IP datagrams with increasing TTL to the destination. The TTL values start from 1. Doubletree modifies the working process of traceroute. Given a source and a destination, a traceroute in Doubletree starts probing with $\text{TTL} = h$, where h is a predefined system parameter. The probing then proceeds toward the destination and backward to the source. Both the backward and forward probings use stop sets. The one for backward probing, called the *local stop set*, consists of all the routers already discovered by the source. The forward probing uses the *global stop set* of $(router, destination)$ pairs accumulated from all sources. A pair enters the global stop set if the router is visited in a traceroute toward the corresponding destination. In either case, the probing stops whenever a member of the stop set is met.

We give an example to show how Doubletree reduces the intermonitor redundancy. Suppose that C and D are two monitors and they will traceroute the paths to the same destination A. Suppose that both C and D set $h = 2$. When C first traceroutes the path to A, C starts from the router h hops away from itself (that is, router 3) and conducts the following:

1. *Forward probing.* C sequentially discovers routers 3 and 1 until it reaches the destination A. In this probing, it accumulates a global stop set of $\{(3,A),(1,A)\}$. This global stop set will be shared with D.

2. *Backward probing.* C discovers router 4 and inserts the router into its local stop set. Now C's local stop set includes all the routers it has discovered, that is, $\{3, 1, 4\}$.

Afterward, when D traceroutes the path to A, D also starts from the router h hops away from itself and conducts the following:

1. *Forward probing.* D first discovers router 3. It checks its global stop set (assuming that C has shared its global stop set with D) and finds that this router has appeared in the global stop set, that is, $(3, A)$. Therefore, D stops forward probing.
2. *Backward probing.* D discovers router 5 and inserts the router into its local stop set.

In this example, router 1 will be visited only once when both C and D traceroute the paths to A. Intermonitor redundancy is hence reduced.

In this paper, we integrate the Doubletree algorithm into Max-Delta to reduce the measurement redundancy. That is, after destination selection by Max-Delta, a traceroute starts and stops under Doubletree's supervision. We study how we can set the parameter h and analyze the reduction in bandwidth consumption through simulations.

3 A Distributed Inference Scheme

In this section, we discuss the design of the distributed inference scheme. We first give an overview of host actions and then explore the tree construction and maintenance issues.

3.1 Design Overview

In the distributed scheme, hosts exchange traceroute results through an overlay tree. Each host maintains a partially discovered topology and uses the Max-Delta heuristic to select traceroute targets by itself. Suppose that A is a new incoming host. We describe the actions of A in a sequential order as follows:

1. *Estimate the network coordinates.* A first uses some tool (for example, GNP or Vivaldi) to estimate its network coordinates. For example, if GNP is used, A should ping a few public landmarks and use the network distances to landmarks and the landmark coordinates to compute its own coordinates.
2. *Join the overlay tree.* A then identifies a host in the system as its parent to join the overlay tree. The detailed tree joining and maintenance mechanisms will be explained later.
3. *Conduct first-round traceroute.* The first-round traceroutes from all the hosts had better form a connected graph among them. Otherwise, the distance between two hosts in the discovered topology D_p may not be available. Therefore, we require A to traceroute the path to its parent in the first round. Clearly, if each host traceroutes like this, all the first-round traceroute paths form a tree spanning the hosts on the overlay.
4. *Distribute the coordinates and first-round traceroute along the tree.* A sends its coordinates and first-round traceroute to its neighbors.
5. A then keeps doing the following things in parallel:
   - *Select paths to be tracerouted.* A maintains a discovered topology based on its own traceroutes and traceroute results from other hosts. Based on that, A can select its traceroute targets as in the naive max-delta. Clearly, it takes $O(V_p \log V_p + E_p + N)$ time to select one traceroute target.
   - *Accept data from its neighbors if any.*
   - *Periodically send data (including its own traceroute results and other hosts' traceroute results) to its neighbors.*

   A aggregates its own traceroute results and data received from its neighbors. It then periodically floods the data along the tree. Clearly, data received from a neighbor are forwarded to all its other neighbors in the tree, and its own traceroute results are sent to all the neighbors.

3.2 Tree Construction

During measurement, each host needs to send traceroute results to others. We hence consider building a source-unspecific tree among hosts. To achieve quick data exchange between hosts, we minimize the diameter of the tree, which is the longest simple path (in terms of the number of overlay hops) in the tree. Previous research has shown that building a minimum-diameter degree-bounded spanning tree is NP-hard, and there are no efficient distributed algorithms to address it [16], [17]. In this paper, we propose a distributed tree construction algorithm based on the outdegree bounds of hosts. In our algorithm, we first identify a host as the tree root, which is usually the host with the largest outdegree bound. We then insert new hosts around the root. During tree construction, we assume that the root is the only source in the system and is about to distribute data to all other hosts. Each new host needs to identify another host as its *parent* in order to join the tree. Except for the parent, all the other neighbors of a host in the tree are called its *children.* Clearly, during real measurement, there is no such parent-child relationship between hosts, since every host is a source.

Each host has an outdegree bound according its edge bandwidth, which indicates how many children a host can have in the tree. The position of a host in the tree depends on its outdegree bound. The larger the outdegree bound that a host has, the closer to the root it is put. In other words, a host has a larger outdegree bound than all its descendants in the tree. Denote B_i as the outdegree bound of host i and denote D_i as the real outdegree of i in the tree. Furthermore, denote $Dist_i$ as the minimum number of overlay hops from i to the root in the tree. Clearly, $Dist_i = Dist_{i's\ parent} + 1$. Finally, we denote New_i as the minimum number of overlay hops from a new host to the tree root if the new host becomes i's descendant. Given a host i, if $B_i > D_i$, $New_i = Dist_i + 1$. Otherwise, $New_i = \min\{New_y \mid \forall y \in i\text{'s children set}\}$.

First, hosts in the tree periodically exchange their $Dist$ and New values with the neighbors. Each host can then accordingly compute its own $Dist$ and New values. When a new incoming host wants to join the tree, it first contacts a public rendezvous point (RP) to obtain the IP address of the root. It then joins the tree, as shown in Algorithm 1. We briefly explain the joining process of a host i as follows: If i is the first joining host, i becomes the root and claims itself to the RP. If this is not the case but i has a larger outdegree bound than the current root, i becomes the new root: It accepts the current root and the children of the current root as its children and then claims itself to the RP. If none of the above cases occurs, i starts a recursive joining procedure from the root. That is, i identifies a host x to start the JOIN(i, x) procedure (see line 14 of Algorithm 1), where x is the root at the beginning. Note that following our algorithm, we have $B_i < B_x$ in the JOIN(i, x) procedure, which is a precondition for invoking JOIN(i, x). Therefore, i can only be x's descendant and cannot be x's ancestor. If x has residual outdegree (that is, $B_x > D_x$), x accepts i as its child. Otherwise, i turns to check x's children. From all x's children whose outdegree bounds are smaller than that of i, the one with the maximum outdegree bound is selected, say, m. If there exists such a host m, i takes up m's position in the tree: i selects m's parent as its parent, accepts m's children as its children, and accepts m as its child. If there is no such a host m (that is, the outdegree bounds of x's children are all larger than or equal to that of i), i has to move one level down. i selects from x's children the host with the smallest New value and repeats the joining process from this host.

Algorithm 1: JOINING PROCEDURE OF HOST i.

```
1: procedure TREEJOIN ($i$)
2:   if $i$ is the first joining host then
3:     $i$ becomes the root and notifies the RP.
4:     return
5:   end if
6:   if $B_i > B_{root}$ then
7:     $i$ becomes the new root: it accepts $root$ and $root$'s
       children as its children and notifies the RP.
8:     return
9:   else
10:    JOIN ($i, root$).
11:    return
12:  end if
13: end procedure

14: procedure JOIN ($i, x$)
15:  if $B_x > D_x$ then
16:    $x$ accepts $i$ as its child.
17:    return
18:  else
19:    if $\exists t \in x$'s children set such that $B_t < B_i$ then
20:      select $x$'s child $m$ such that $B_m = \max\{B_y | \forall y \in x$'s
         children set & $B_y < B_i\}$.
21:      $i$ replaces $m$ (that is,
         $i$ selects $m$'s parent as its parent and accepts $m$'s
         children as its children) and accepts $m$ as its child.
22:      return
23:    else
24:      select $x$'s child $n$ such that $New_n = \min\{New_y | \forall y \in x$'s
         children $set\}$.
25:      JOIN ($i, n$).
26:      return
27:    end if
28:  end if
29: end procedure
```

We show an example in Fig. 2, where hosts A, B, C, D, E, F, G, and H join the tree in a sequential order. We describe the joining steps (corresponding to the figures in Fig. 2) as follows:

(a) When A joins the tree, it first contacts the RP. Since it is the first joining host in the tree, A becomes the root and claims itself to the RP.
(b) When B joins the tree, it also contacts the RP and knows that A is the tree root. B then contacts A to compare their outdegree bounds.
(c) B finds that its own outdegree bound is larger than that of A (that is, $B_B > B_A$). Hence, B becomes the new root and accepts A as its child.
(d) When C joins the tree, it first contacts the RP and then contacts the tree root B.
(e) C finds that its own outdegree bound is smaller than that of B. It becomes B's child, since B still has a residual outdegree.
(f) D, E, and F join the tree in a similar way as C. All these hosts have smaller outdegree bounds than the root B.
(g) When G joins the tree, it contacts the RP and the tree root B. Now, B does not have a residual outdegree and cannot accept G as its child. B finds that it has a child (that is, E) whose outdegree bound is smaller than G. B hence forwards G to this child.
(h) G replaces E and accepts E as its child.
(i) When H joins the tree, it contacts the RP and the tree root B. Still, B cannot accept H as its child. Now, all of B's children have larger outdegree bounds than H. B hence forwards H to its child with the minimum New value, which is A in this example (in fact, A, C, D, G, and F have the same New value of 2).
(j) H becomes A's child.

Note that in our scheme, data exchanged between two hosts are of a small size. For example, a traceroute result is of several kilobytes. In view of the capabilities of today's computers and networks, a powerful PC can support simultaneous data exchange with hundreds of neighbors. The outdegree bounds of hosts in our scheme are hence much larger than those in other applications such as streaming or file sharing.

3.3 Tree Maintenance

Upon leaving or the failure of a host, all its children in the tree need to rejoin the tree. A rejoining process is similar to host joining.

As well known, tree-based overlay is not resilient to host dynamics, as host failure or leaving affects all its descendants in the tree. A tree hence does not perform well for streaming applications [32]. This is because in streaming, each host needs to continuously receive data at a certain rate. During rejoining, a host cannot receive any data and has to suffer service outage. The loss rate then increases, and the streaming quality decreases. Therefore, in streaming applications, there have been many proposals to reduce the time for host

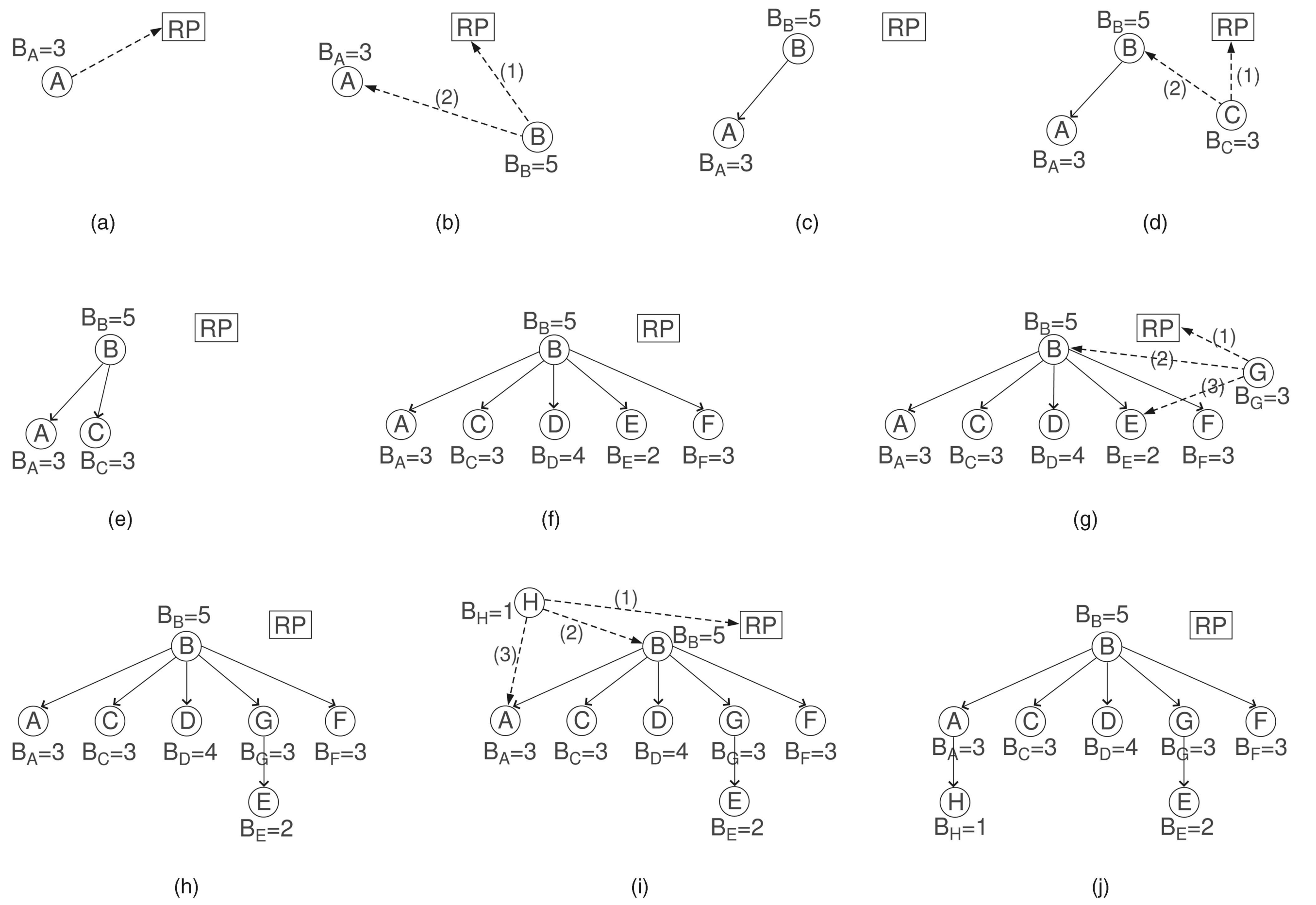


Fig. 2. An example of tree construction (for a certain host x, B_x is the outdegree bound of x).

rejoining [33]. However, in our inference scheme, host rejoining does not have high impact on system performance. Topology inference does not require continuous data receiving at hosts. When a host is disconnected from the tree, it can still select traceroute targets based on its own topology and keep tracerouting. Therefore, in our scheme, we do not need to focus on reducing the host rejoining time as in streaming applications.

Certainly, frequent host joining and leaving decrease the stability of the tree. We may integrate previous research to improve the tree stability. For example, it has been shown that the longest first tree is often more stable, where a new host selects from all its parent candidates the longest lived one as the parent [34], [35]. We can then take the host lifetime into consideration when constructing the tree.

4 Reducing Measurement Cost

In this section, we propose several techniques to reduce the measurement cost for topology inference, including reducing measurement redundancy, reducing traceroute size, and reducing computational overhead.

4.1 Reducing Measurement Redundancy

We integrate Doubletree into Max-Delta to reduce the measurement redundancy. That is, after a host selects a traceroute target by Max-Delta, it starts and completes the traceroute under Doubletree's supervision. In the distributed inference scheme, all traceroute results will be distributed to all the hosts in the system. The sharing of the global stop set is hence trivial. Now, we focus on the selection of h.

The selection of h is important to the system performance. According to [19], a small h leads to high intramonitor redundancy, whereas a large h leads to high intermonitor redundancy. In [19], each monitor sets its own value of h in terms of the probability p that the path between the monitor and a random destination does not have more than h hops. For example, Fig. 3 shows the cumulative mass function of p for a Skitter monitor (see [19, Fig. 5]). In order to restrict the probability that the first ICMP message of a traceroute (with $\mathrm{TTL} = h$) hits the destination to a certain value, say, 10 percent, the monitor should start probing at $h = 10$ hops. In [19], Donnet et al. suggest setting p to 0.05. They also suggest estimating the distribution of p by sampling a few randomly selected paths. However, this estimation method is not feasible here. This is because the estimation accuracy is highly related to the sampling size. However, in Max-Delta, it is impossible to afford a large sampling size, because the number of traceroutes conducted by each host is small (for example, 10-14 in a group of 256 hosts), and it is not feasible to have a larger sampling cost than the measurement cost.

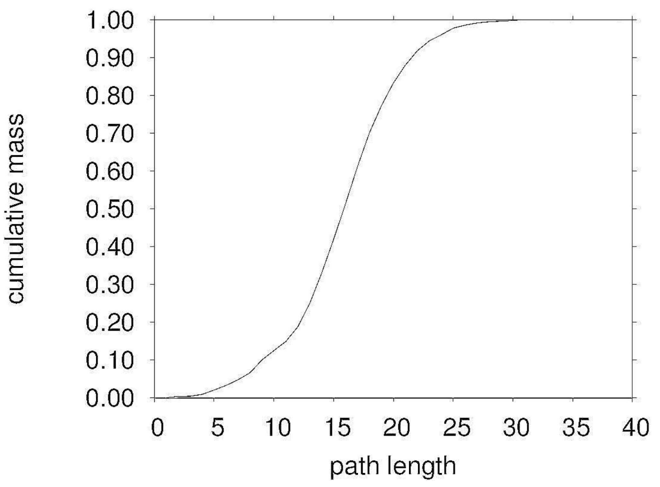


Fig. 3. Length of paths for a Skitter monitor (from [19]).

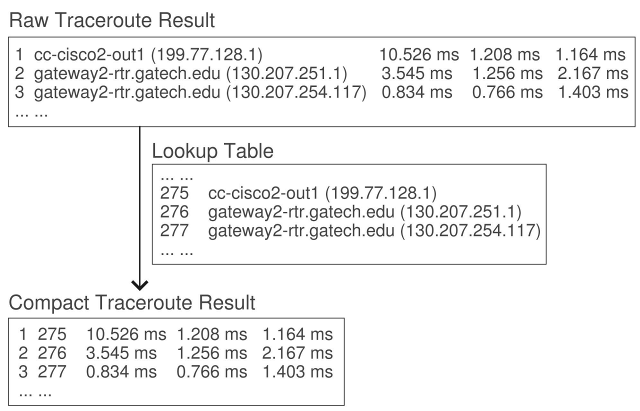
We note that Max-Delta can discover a majority of the underlay topology in its first several iterations. Table 1 shows the link ratio (as defined in Section 5.1) achieved by Max-Delta in the first several iterations on Transit-Stub topologies. In a group of 256 hosts, 74.3 percent of underlay links can be discovered after the first iteration of traceroutes. After 13 iterations, over 97.1 percent of links are discovered. When $N = 1,024$, 65.1 percent of links are discovered after the first iteration. After 19 iterations, over 91.1 percent of links are discovered. We hence estimate h as follows: Since each host maintains a partially discovered topology, a host can compute the shortest paths between itself and all the other hosts in the topology. When the discovered topology contains the majority of the links on the underlay (this may be the case after several iterations), the shortest path between two hosts in the discovered topology is supposed to approximate the actual path between them. Note that the partially discovered topology is not complete yet. Missing of links clearly leads to the overestimation of path length in the discovered topology. We hence discard a certain number of paths with the most hops and use the rest of the results to compute the distribution of p. Our simulation results show that it is good to discard around 20 percent to 30 percent paths with the most hops.

4.2 Reducing Traceroute Size

In a traceroute result, a router is represented by its IP address and router name, which often consist of 20-40 digits or letters. On the other hand, a router is often visited multiple times in different traceroutes, even with Doubletree integration (as shown in Section 5.3). We hence consider using a compact form to represent routers in order to reduce the size of traceroute results.

TABLE 1
Link Ratio Achieved by Max-Delta in Different Iterations (in Percent)

Iteration	1	3	5	7	9	11	13	15	17	19
$N = 256$	74.3	82.3	86.7	91.8	94.8	96.8	97.1	97.4	97.8	98.1
$N = 1024$	65.1	74.2	83.3	85.2	86.2	87.8	88.4	89.0	90.2	91.1



Raw Traceroute Result
```
1  cc-cisco2-out1 (199.77.128.1)              10.526 ms  1.208 ms  1.164 ms
2  gateway2-rtr.gatech.edu (130.207.251.1)     3.545 ms  1.256 ms  2.167 ms
3  gateway2-rtr.gatech.edu (130.207.254.117)   0.834 ms  0.766 ms  1.403 ms
... ...
```

Lookup Table
```
... ...
275  cc-cisco2-out1 (199.77.128.1)
276  gateway2-rtr.gatech.edu (130.207.251.1)
277  gateway2-rtr.gatech.edu (130.207.254.117)
... ...
```

Compact Traceroute Result
```
1 275  10.526 ms  1.208 ms  1.164 ms
2 276   3.545 ms  1.256 ms  2.167 ms
3 277   0.834 ms  0.766 ms  1.403 ms
... ...
```

Fig. 4. Setting up a lookup table for routers.

A straightforward approach is to discard router names and only keep router IPs in traceroute results. As each IP can uniquely identify an interface of a router, discarding the router names does not introduce additional ambiguity. However, an IP address still contains around 12 digits or letters. We would like to further reduce the representation size. Furthermore, it is better to keep the router name information, which may be used in future studies. For example, router names can be used to infer the geographic locations of routers [36].

Therefore, we set up a mapping table to map each router to an integer. First, such mapping should be one to one so that traceroute results can be freely transformed between two forms without ambiguity. Second, the mapping should be universal to all the hosts. Therefore, we identify one host in the tree to conduct mapping. A possible choice is the tree root. The root can maintain a lookup table for routers. When the root finds a new router in traceroute results, it inserts the router into the lookup table. It then replaces routers in traceroute results by their corresponding serial numbers in the lookup table. In this way, each router is represented by an integer, whose length depends on the total number of routers (not more than six digits in practice). Later on, the root delivers the compact traceroute results and the lookup table to other hosts. Note that the lookup table is only delivered once to each host. Fig. 4 shows an example of a raw traceroute result and a compact traceroute result.

In the above approach, the root always receives raw traceroute results without compression. To reduce bandwidth consumption at the root, we can accordingly decrease the number of children of the root. If a root leaves or fails, according to the tree construction mechanism in Section 3, a new root will be selected from the root's children as the one with the largest outdegree bound. The new root then takes the responsibility of compressing traceroute results. We may also use the root replication technique proposed in [37] to improve the root resilience. In the technique, the root has multiple backup hosts. All the information at the root is copied at each of the backup hosts. If the current root fails or leaves, one of the backup hosts serves as the new root.

4.3 Reducing Computational Overhead

We propose two techniques to reduce the computational overhead of the inference scheme.

4.3.1 Topology Abstraction

Given an underlay topology, we can simplify it before computing the shortest paths between hosts. That is, we consider reducing V_p and E_p in the topology. For example, in Fig. 1, links 3-4 and $4-C$ can be concatenated as one link, and the length of the new link is equal to the length of link 3-4 plus the length of link $4-C$. This is because any path crossing link 3-4 also crosses link $4-C$. After concatenating them, the distance of the shortest path between any two hosts in the new topology is the same as that in the original topology.

Define a *path segment* as a maximal subpath of an overlay path so that all the inner routers in the subpath are not adjacent to any other physical links in the underlay network. In other words, two end points of a path segment have degree 1 or not less than 3 in the underlay network, whereas all its inner routers have degree 2. In the following, we call the inner router in a path segment a *segment router* and the link adjacent to a segment router a *segment link*.

Given an underlay topology, we would like to find all the path segments in it and replace each of them with a single virtual link. We can use any traverse algorithm such as Breadth-First Search or Depth-First Search to visit all the links and routers in the topology. Whenever we find a router of degree 2, we merge its two adjacent links. By traversing the whole topology in $O(V_p + E_p)$ time, all the path segments can be abstracted as single links. We can then use this simplified topology as the basis to compute the shortest paths between hosts.

4.3.2 Reducing Computational Frequency

To select a traceroute target, a host needs to compute the distance gaps along all unmeasured paths adjacent to it. In fact, after each computing, we can reuse the results to select the subsequent targets without computing again. Suppose that a host A has computed all the Δ values at a certain time. A marks all the links in the discovered topology as *old*. In the following, it marks the newly discovered links as *new*. Only when the number of new links is larger than a certain threshold t does A recompute the Δ values between itself and other hosts and remark all the links as old.

As shown, a large portion of the topology can be discovered in the first several iterations. Later on, only a few new links are discovered in one iteration. As the new links are few, they have limited impact on the distance gaps, and we do not need to compute the distance gaps in each iteration. The computational overhead is hence reduced.

5 Illustrative Numerical Results

In this section, we evaluate our inference scheme through simulations on Internet-like topologies and measurements on PlanetLab.

5.1 Simulation Setup

We generate a number of *Transit-Stub* topologies with GT-ITM [38]. Each topology is a two-layer hierarchy of transit networks and stub networks. Unless otherwise indicated, a topology contains 3,200 routers and around 20,000 links. We randomly put N hosts into the network ($N = 500$, unless otherwise indicated). Each host is connected to a unique stub router with a 1-ms delay, whereas the delay of core links is given by the topology generator.

Furthermore, we randomly select 79 hosts from PlanetLab and conduct all-pair traceroutes between them [39]. Due to network and host dynamics (some hosts unexpectedly failed during our measurements), a small portion of the traceroutes cannot be completed. The resulting topology contains 5,589 overlay paths (out of the total $78 \times 79 = 6,162$ ones), 1,950 links, 946 known routers, and some anonymous routers.

For a host in the system, the receiving of traceroute results from other hosts and the conducting of traceroutes are in parallel. In our simulations, for simplicity, we assume that data exchange between hosts is very quick and the conducting of one traceroute is considerably slow. In other words, we use similar settings as in the centralized Max-Delta; that is, in each iteration, a host conducts one traceroute and receives one traceroute result from each of the other hosts. This assumption does not qualitatively affect the results and conclusions in the following. Based on this assumption, our distributed inference scheme has the same measurement efficiency as the centralized Max-Delta scheme. Since the performance of the centralized Max-Delta has been thoroughly evaluated in [13] and [15], we do not repeatedly present similar results here.

We have done simulations on both types of topologies. We find that the conclusions on the PlanetLab topology are qualitatively the same as those on the Transit-Stub topologies. We hence mainly present results on Transit-Stub topologies. We use the following metrics to evaluate an inference scheme:

1. *Router visiting frequency* ϕ. This is defined as the number of occurring times of a router in a set of traceroute paths.
2. *Link ratio* β. This is defined as the ratio of the number of links in the inferred topology to the total number of links in the actual underlay topology [15].
3. *Router ratio* γ. This is defined as the ratio of the number of routers in the inferred topology to the total number of routers in the actual underlay topology [15].
4. *Resource usage.* Given a traffic between two points, the resource usage is computed as the size of packets delivered times the delay between the two points. We compute the resource usage of an inference scheme as the total resource usage of traceroutes in the inference.
5. *ICMP message reduction ratio* θ. This is defined as the ratio of the number of ICMP messages reduced by Doubletree to the total number of ICMP messages by naive traceroutes.

5.2 Performance of Distributed Inference

As mentioned, the measurement efficiency (in terms of link ratio and router ratio) of the distributed scheme is similar to the centralized Max-Delta. We hence skip it. Fig. 5 shows the tree diameter versus the group size. The outdegree bounds of hosts are uniformly distributed within [30, 100]. As mentioned, the data exchanged are of a small size, and hosts can have relatively large outdegree bounds. In the figure, we see that the tree diameter is kept low. When

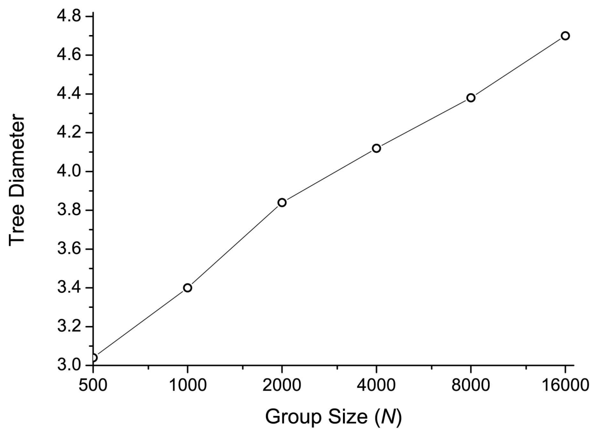


Fig. 5. Tree diameter versus group size.

$N = 16{,}000$, the tree diameter is only 4.7. Therefore, data exchange between hosts can be quickly accomplished.

The consumption of edge bandwidth at a host mainly consists of three parts:

1. *Bandwidth for traceroute measurements.* The host may actively traceroute some paths. The host may also be selected as the traceroute target by other hosts. It hence consumes bandwidth for sending and receiving ICMP messages.
2. *Bandwidth for the exchange of traceroute results.* Ideally, a traceroute result is distributed from the traceroute source to all the other hosts. In a group of N hosts, it takes, in total, $2(N-1)s$ edge bandwidth at hosts to distribute the traceroute result (including data receiving and forwarding), where s is the size of the traceroute result. Therefore, during the whole inference, the average edge bandwidth consumed for data exchange at a host is $2(N-1)S/N$, where S is the total size of traceroute results. The value of S is shown in Fig. 7b.
3. *Bandwidth for tree construction and maintenance.* The host needs to exchange control messages with other hosts to join and maintain the overlay tree.

As compared to the centralized Max-Delta, parts 2 and 3 are additional bandwidth consumption. On the other hand, in the centralized Max-Delta, each host needs to send its traceroute results to the central server. This incurs, on the average, S/N bandwidth consumption at each host (we do not count the bandwidth consumption at the server). Therefore, in terms of bandwidth assumption for the exchange of traceroute data, the distributed scheme incurs an $2(N-1)S/N - S/N = (2N-3)S/N$ additional bandwidth consumption at each host.

5.3 Reducing Measurement Redundancy

We first test the selection mechanisms of h on different topologies. We generate four types of Transit-Stub topologies with different network sizes. The number of routers (or links) in the topologies are 1,800, 3,200, 5,120, and 12,000 (or around 9,000, 20,000, 32,000, and 170,000), respectively. In the following, these four types of topologies are denoted as R-1800, R-3200, R-5120, and R-12000, respectively. For each type, we generate five topologies. We then randomly put 500 hosts into the network and compute the average number of hops in the interhost paths. After repeating the above process for 10 times, the average numbers of hops in these four types of topologies are 7.5, 8.9, 9.7, and 14.2, respectively. We test three selection mechanisms of h. The first one sets h to a constant 4. The second one estimates h as in Doubletree, with a sampling of 20 paths. The last one estimates h as in Section 4.1, where 25 percent of paths with the most hops are discarded. These three methods are denoted as "*Constant*," "*Sampling*," and "*Topology-based*," respectively.

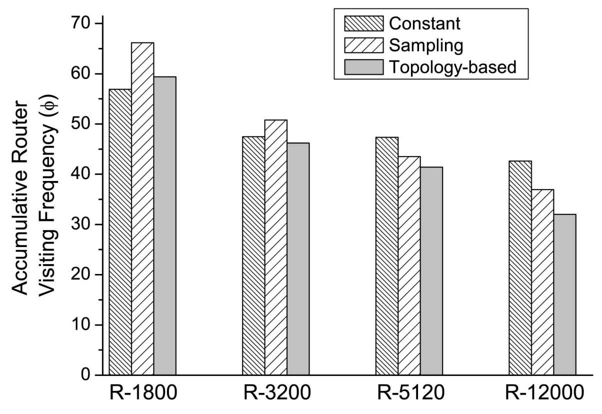
Fig. 6a shows ϕ at the 30th iteration achieved by different h-selection mechanisms. On R-1800 and R-3200, *Constant* has good performance. However, it does not perform well on R-5120 and R-12000. It shows that *Constant* is not flexible to different networks. *Sampling* always achieves higher ϕ than the *Topology-based* approach. It also incurs additional sampling cost. It is hence not a good choice. The *Topology-based* approach achieves good performance on all the four types of topologies. It does not incur any additional cost and is hence applicable for our inference scheme.

Figs. 6b, 6c, 6d, 6e, and 6f show the performance of the inference scheme with Doubletree integration on R-3200, where $N = 500$. Fig. 6b shows the accumulative link ratios β achieved by the inference schemes. The naive Max-Delta performs slightly better than the integration one. As the iteration number increases, β achieved by Max-Delta converges toward 0.992, whereas that of the integration scheme converges toward 0.965. On the other hand, these two schemes achieve almost the same router ratio, as shown in Fig. 6c. These results show that the stop rules in Doubletree have slight impact on the measurement accuracy.

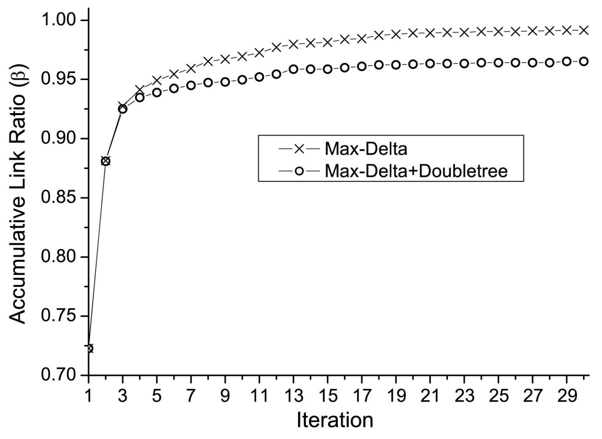
Fig. 6d compares the accumulative router visiting frequencies achieved by the schemes. In Max-Delta, ϕ quickly increases with the iteration number. In the first iteration, ϕ is only 4.9. However, after 30 iterations, the average ϕ increases to 98.5. On the other hand, in our PlanetLab measurements, there are totally 82,212 known routers in the 5,589 paths. However, there are actually 946 different routers. It means that each router, on the average, appears $82{,}212/946 = 86.9$ times. This confirms that routers are repeatedly visited for many times. The integration of Doubletree can significantly reduce ϕ. After 30 iterations in the integration scheme, the average ϕ is only 47.1, which is less than half of 98.5.

Fig. 6e shows the resource usage for traceroutes in the integration scheme. The flat curve in the figure indicates the resource usage for measurement in the integration scheme, and the precipitous curve indicates the reduction in resource usage due to Doubletree. Clearly, the sum of these two values is the resource usage for naive traceroutes. As shown, with Doubletree, the resource usage for measurement increases slowly. Furthermore, the amount of reduction increases quickly when the iteration continues. It shows that Doubletree can significantly reduce the resource usage, especially when the number of iterations is large.

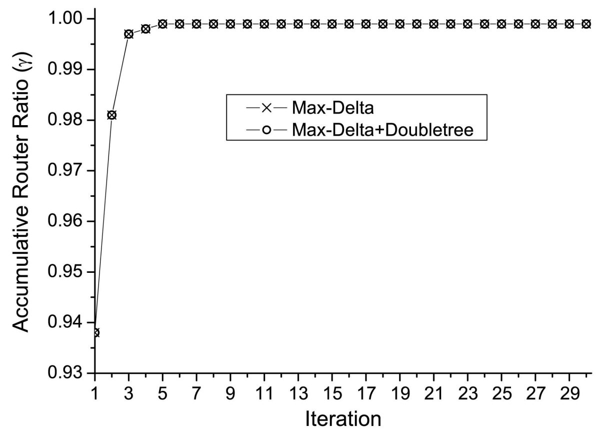
Fig. 6f shows the ICMP message reduction ratio θ in the integration scheme. The curve denoted "Per iteration" shows the result in each individual iteration. Note that in the second iteration, θ is negative. This is because the probing distance h is too large, and the first ICMP message of a traceroute has reached the destination. As a result, more ICMP messages are sent out than that in a naive traceroute. In



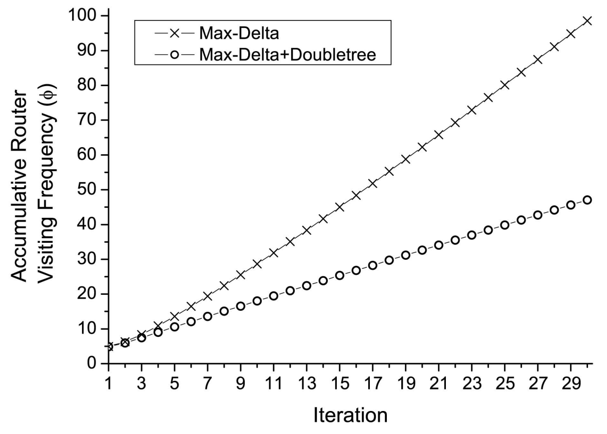
(a)



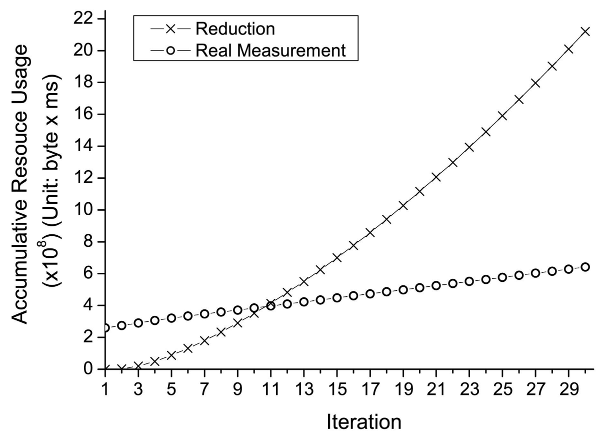
(b)



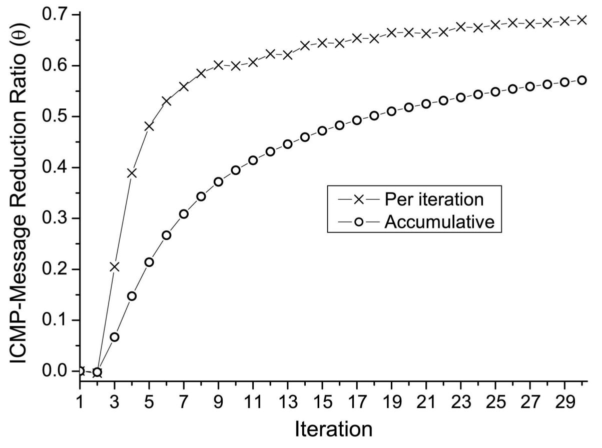
(c)



(d)



(e)



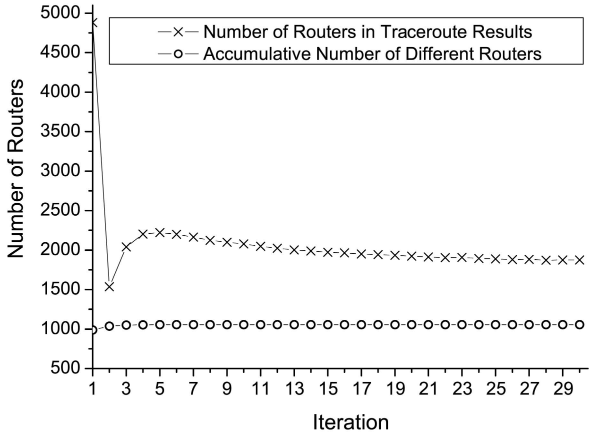
(f)

Fig. 6. Reducing the measurement redundancy $(N = 500)$. (a) Selection of h. (b) Accumulative link ratio. (c) Accumulative router ratio. (d) Accumulative router visiting frequency. (e) Accumulative resource usage. (f) ICMP message reduction ratio.

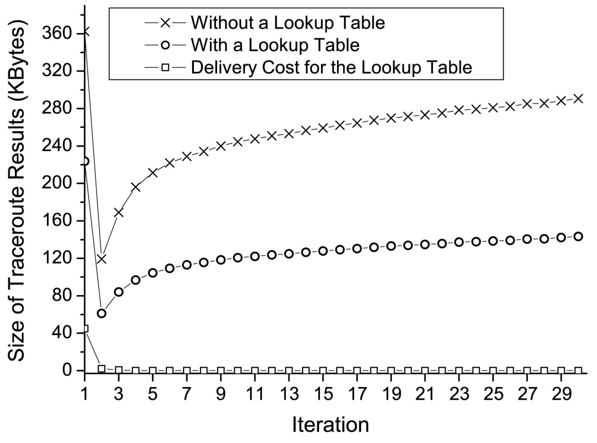
the following iterations, θ quickly increases to around 0.6. It shows that a significant reduction in the number of ICMP messages has been achieved. The curve denoted "Accumulative" shows the accumulative ICMP message reduction ratio. After 30 iterations, we can achieve an overall 57 percent reduction in the number of ICMP messages. Clearly, the integration of Doubletree is efficient and effective.

5.4 Reducing Traceroute Size

As shown in Fig. 6d, the router visiting frequency is remarkably high, even with the integration of Doubletree. A lookup table for routers can hence significantly reduce the traceroute size. Fig. 7a shows the number of routers discovered in the integrated scheme combining Max-Delta and Doubletree. The upper curve shows the number of



(a)



(b)

Fig. 7. Reducing the traceroute size ($N = 500$ on R-3200). (a) Number of routers. (b) Size of traceroute results.

routers reported in each iteration. In each iteration, each host usually traceroutes one path (in some cases, a host may not be able to find a traceroute target and hence does not conduct traceroute). In the first iteration, each host tracerouts its parent in the tree, and the total number of routers is 4,899. It means that a path, on the average, contains $4{,}899/500 = 9.8$ routers. In the next iteration, the total number of routers is only 1,534. This is because Max-Delta preferentially selects a path with a large Δ value, which corresponds to a small $euclidean$ distance and, hence, a short path. In other words, Max-Delta preferentially selects the shortest paths. Therefore, in the following iterations, we see an increase in the total number of routers reported. The bottom curve shows the accumulative number of different routers achieved by the scheme. In the first four iterations, over 99 percent of routers have been discovered. It shows that routers are more easily and quickly discovered than links (refer to Figs. 6b and 6c). Since the number of different routers is small, a router can be uniquely represented by a small integer in a lookup table.

We compare the sizes of traceroute results with and without the lookup table in Fig. 7b. The size of traceroute results is estimated as follows: The formats of raw traceroute results and compact traceroute results follow those in Fig. 4. From the PlanetLab measurements, we have 946 different routers. The representation of a router (including the router name and router IP), on the average, consists of 41.5 letters or digits. With a lookup table, each router can be represented by four digits (four digits can represent at most 10,000 routers). In the figure, the upper curve shows the results with no lookup table, and the middle curve shows the results with a lookup table (including the delivery cost for the lookup table). As shown, the lookup table can, on the average, reduce the traceroute size by 50.2 percent. In addition, we show the delivery cost for the lookup table in different iterations. In the first iteration, almost the whole lookup table has been constructed. The lookup table is around 45 Kbytes. In the following iterations, new entries of the lookup table are very few, and their size is not more than 2.2 Kbytes. This is because the first-round traceroutes can discover over 93.7 percent of routers (as shown in Fig. 7a). In the following iterations, there are only a few new routers discovered. As the lookup table is delivered to each host once, it does not incur high delivery overhead. In summary, the use of a lookup table is efficient and effective.

5.5 Reducing Computational Overhead

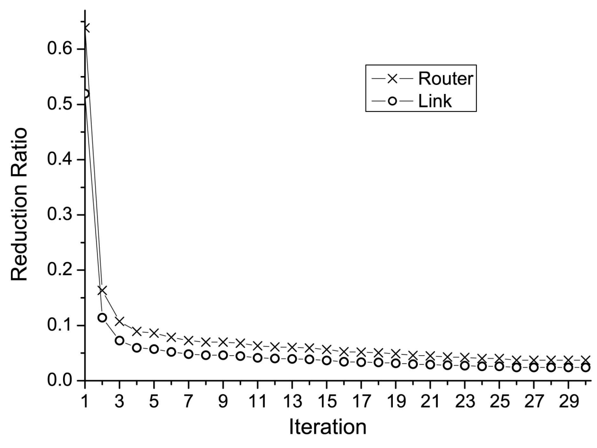
Fig. 8 shows the reduction in the computational overhead on R-3200. Fig. 8a shows the ratio of the number of segment routers (or segment links) to the number of all routers (or all links) in different iterations. In the first several iterations, there are lots of segment routers and segment links in the inferred topology. As the iteration continues, such ratios converge to around 2 percent to 4 percent. Clearly, topology abstraction can significantly reduce the computational complexity in the first several iterations. As topology abstraction incurs additional computational overhead, we may only conduct it in the first several iterations.

On the other hand, if the computational frequency is reduced as proposed, only a few iterations require the distance gap computing (mostly in the first several iterations). Fig. 8b shows the result with $t = 3$ percent; that is, the distance gap computing is conducted whenever the number of new links is equal to or large than 3 percent of the number of old links. As shown, in the first 30 iterations, only the 1st, 2nd, 4th, and 18th iterations require the computing (as indicated by the vertical lines). In addition, the resultant link ratio is slightly lower that in a naive integration scheme. Therefore, frequency reducing has small impact on the measurement accuracy.

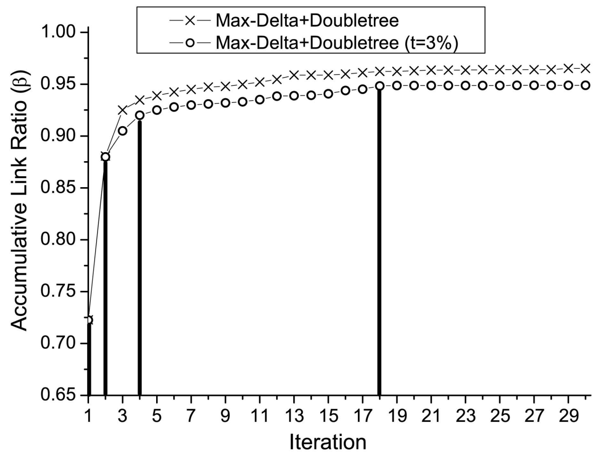
As shown, by the frequency reducing technique, most computational loads for distance gaps are in the first several iterations. As topology abstraction can significantly simplify the topology in the first several iterations, we can combine the two techniques to efficiently reduce the computational overhead at a host.

6 Conclusion

Previously, Max-Delta has been proposed to infer a highly accurate topology among a group of hosts with a low number of traceroutes. As it is centralized and not scalable, we propose in this paper a distributed inference scheme for



(a)



(b)

Fig. 8. Reducing the computational overhead ($N = 500$ on R-3200). (a) Router and link reduction by topology abstraction. (b) Reducing the computational frequency (the vertical lines indicate the iterations that require the computing of distance gaps).

scalable topology inference. In our scheme, hosts form an overlay tree to exchange traceroute results. A host independently selects paths for tracerouting, without the need of central scheduling. We further propose several techniques to reduce the measurement cost, including reducing measurement redundancy, reducing traceroute size, and reducing computational overhead. Simulation results show that our data delivery tree has a low diameter and that the proposed improvement mechanisms can significantly reduce bandwidth consumption and computational overhead for measurements.

Acknowledgments

The work at HKUST was supported, in part, by the Research Grant Council of the Hong Kong Special Administrative Region under Grant HKUST611107 and Hong Kong Innovation and Technology Commission under Grant GHP/045/05. The work of W. Tu was supported by the Embark Postdoctoral Fellowship of Ireland under Grant 501-et-5044890.

References

[1] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "Resilient Overlay Networks," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01),* pp. 131-145, Oct. 2001.
[2] Y.H. Chu, S. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE J. Selected Areas in Comm.,* vol. 20, no. 8, pp. 1456-1471, Oct. 2002.
[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. ACM SIGCOMM '02,* pp. 205-217, Aug. 2002.
[4] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses," *Proc. ACM SIGCOMM '05,* pp. 73-84, Aug. 2005.
[5] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, J. Hellerstein, and S. Shenker, "A Case Study in Building Layered DHT Applications," *Proc. ACM SIGCOMM '05,* pp. 97-108, Aug. 2005.
[6] D.G. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan, "Topology Inference from BGP Routing Dynamics," *Proc. ACM Internet Measurement Workshop (IMW '02),* pp. 243-248, Nov. 2002.
[7] F. Wang and L. Gao, "On Inferring and Characterizing Internet Routing Policies," *Proc. ACM Internet Measurement Workshop (IMW '03),* pp. 15-26, Oct. 2003.
[8] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum Likelihood Network Topology Identification from Edge-Based Unicast Measurements," *Proc. ACM SIGMETRICS '02,* pp. 11-20, 2002.
[9] M. Coates, A. Hero, R. Nowak, and B. Yu, "Internet Tomography," *IEEE Signal Processing Magazine,* vol. 19, no. 3, pp. 47-65, May 2002.
[10] *Traceroute,* http://www.traceroute.org/, 2007.
[11] M. Kwon and S. Fahmy, "Topology-Aware Overlay Networks for Group Communication," *Proc. 12th ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '02),* pp. 127-136, May 2002.
[12] J. Han, D. Watson, and F. Jahanian, "Topology Aware Overlay Networks," *Proc. IEEE INFOCOM '05,* pp. 2554-2565, Mar. 2005.
[13] X. Jin, Y. Wang, and S.-H.G. Chan, "Fast Overlay Tree Based on Efficient End-to-End Measurements," *Proc. IEEE Int'l Conf. Comm. (ICC '05),* pp. 1319-1323, May 2005.
[14] X. Jin, Q. Xia, and S.-H.G. Chan, "A Cost-Based Evaluation of End-to-End Network Measurements in Overlay Multicast," *Proc. IEEE INFOCOM Mini-Symposium '07,* June 2007.
[15] X. Jin, W.-P.K. Yiu, S.-H.G. Chan, and Y. Wang, "Network Topology Inference Based on End-to-End Measurements," *IEEE J. Selected Areas in Comm.,* vol. 24, no. 12, pp. 2182-2195, Dec. 2006.
[16] S.Y. Shi, J.S. Turner, and M. Waldvogel, "Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks," *Proc. 11th ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '01),* pp. 83-91, 2001.
[17] S.Y. Shi and J.S. Turner, "Routing in Overlay Multicast Networks," *Proc. IEEE INFOCOM '02,* pp. 1200-1208, June 2002.
[18] B. Donnet, T. Friedman, and M. Crovella, "Improved Algorithms for Network Topology Discovery," *Proc. Sixth Int'l Workshop Passive and Active Network Measurement (PAM '05),* Mar. 2005.
[19] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient Algorithms for Large-Scale Topology Discovery," *Proc. ACM SIGMETRICS '05,* pp. 327-338, June 2005.
[20] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Deployment of an Algorithm for Large-Scale Topology Discovery," *IEEE J. Selected Areas in Comm.,* vol. 24, no. 12, pp. 2210-2220, Dec. 2006.
[21] *Skitter,* http://www.caida.org/tools/measurement/skitter/, 2007.
[22] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet Map Discovery," *Proc. IEEE INFOCOM '00,* pp. 1371-1380, Mar. 2000.
[23] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," *Proc. ACM SIGCOMM '02,* pp. 133-145, Aug. 2002.
[24] P. Barford, A. Bestavros, J. Byers, and M. Crovella, "On the Marginal Utility of Network Topology Measurements," *Proc. ACM Internet Measurement Workshop (IMW '01),* pp. 5-17, Nov. 2001.
[25] A. Broido and K. Claffy, "Internet Topology: Connectivity of IP Graphs," *Proc. SPIE Int'l Conf. and Exhibits on the Convergence of IT and Comm. (ITCom '01),* Aug. 2001.

[26] B. Yao, R. Viswanathan, F. Chang, and D.G. Waddington, "Topology Inference in the Presence of Anonymous Routers," *Proc. IEEE INFOCOM '03,* pp. 353-363, Apr. 2003.

[27] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. ACM SIGCOMM '99,* pp. 251-262, Sept. 1999.

[28] T.S.E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," *Proc. IEEE INFOCOM '02,* pp. 170-179, June 2002.

[29] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," *Proc. ACM SIGCOMM '04,* pp. 15-26, Aug. 2004.

[30] X. Jin, Q. Xia, and S.-H.G. Chan, "A Distributed Approach to End-to-End Network Topology Inference," *Proc. IEEE Int'l Conf. Comm. (ICC '07),* June 2007.

[31] X. Jin, W.-P.K. Yiu, and S.-H.G. Chan, "Improving the Efficiency of End-to-End Network Topology Inference," *Proc. IEEE Int'l Conf. Comm. (ICC '07),* June 2007.

[32] X. Zhang, J. Liu, B. Li, and T.-S.P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," *Proc. IEEE INFOCOM '05,* pp. 2102-2111, Mar. 2005.

[33] Z. Fei and M. Yang, "A Proactive Tree Recovery Mechanism for Resilient Overlay Multicast," *IEEE/ACM Trans. Networking,* vol. 15, no. 1, pp. 173-186, Feb. 2007.

[34] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points," *Proc. ACM SIGCOMM '04,* pp. 107-120, Aug. 2004.

[35] G. Tan and S.A. Jarvis, "Improving the Fault Resilience of Overlay Multicast for Media Streaming," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 6, pp. 721-734, June 2007.

[36] V.N. Padmanabhan and L. Subramanian, "An Investigation of Geographic Mapping Techniques for Internet Hosts," *Proc. ACM SIGCOMM '01,* pp. 173-185, Aug. 2001.

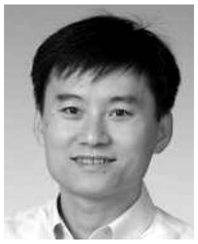
[37] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," *Proc. Fourth Symp. Operating System Design and Implementation (OSDI '00),* pp. 197-212, Oct. 2000.

[38] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proc. IEEE INFOCOM '96,* pp. 594-602, Mar. 1996.

[39] *PlanetLab,* http://www.planet-lab.org, 2007.

Xing Jin received the BEng degree in computer science and technology from Tsinghua University, Beijing, in 2002. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon. Since 2006, he has been a junior editor of the *Journal of Multimedia.* His research interests include overlay multicast, and its applications and QoS issues, Internet topology inference, end-to-end measurements, and peer-to-peer streaming. He is a student member of the IEEE Computer Society. He received the Microsoft Research Fellowship in 2005.

Wanqing Tu received the PhD degree from the City University of Hong Kong in 2006. She is currently a postdoctoral researcher of computer science at the University College Cork, Ireland. Her research interests include QoS, overlay networks, wireless mesh networks, end-host multicast, and distributed computing. She received the Embark Postdoctoral Fellowship of Ireland and the Best Paper Award in the 2005 International Conference on Computer Networks and Mobile Computing (ICCNMC). She is a member of the IEEE Computer Society.



S.-H. Gary Chan received the BSE degree (with highest honors) in electrical engineering from Princeton University, Princeton, New Jersey, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems, and the MSE and PhD degrees in electrical engineering from Stanford University, Stanford, California, in 1994 and 1999, respectively, with a minor in business administration. He is currently an associate professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, and an adjunct researcher at Microsoft Research Asia, Beijing. From 1998 to 1999, he was a visiting assistant professor of networking with the Department of Computer Science, University of California, Davis. From 1992 to 1993, he was a research intern at the NEC Research Institute, Princeton. From 1993 to 1994, he was a William and Leila Fellow at Stanford University. From 2003 to 2006, he was the vice-chair of the IEEE ComSoc Multimedia Communications Technical Committee (MMTC). He is a guest editor of the *IEEE Communications Magazine* (special issue on peer-to-peer multimedia streaming) and the *Springer Multimedia Tools and Applications* (special issue on advances in consumer communications and networking). He was the cochair of the Workshop on Advances in Peer-to-Peer Multimedia Streaming for the 15th ACM International Conference on Multimedia (Multimedia 2005), the 2006 IEEE GLOBECOM Multimedia Symposium, and the 2005 IEEE ICC Multimedia Symposium. He is a cochair of the 2007 IEEE ICC Multimedia Symposium. His research interests include multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks. He is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa and a senior member of the IEEE Computer Society. He is the recipient of the Charles Ira Young Memorial Tablet and Medal from the Princeton University and the 1993 POEM Newport Award of Excellence.