Serving Dynamic Groups in Application-Level Multicast

Xing Jin Wan-Ching Wong S.-H. Gary Chan
Department of Computer Science,
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon Hong Kong
Email: {csvenus, wwilliam, gchan}@cs.ust.hk
Tel: +852 2358-6990 Fax: +852 2358-1477

Abstract— **We propose an ALM framework for dynamic groups (such as stock-quote application) in this paper, where users may hop from one multicast group to another quite frequently even though the total pool of users in the system may remain quite stable. Our approach efficiently maintains multiple multicast trees for dynamic subsets of end-hosts, and hence is called subset-ALM (SALM).**

SALM first builds a relatively stable mesh consisting of all end-hosts for control messaging, which is used to efficiently guide the construction of dynamic overlay trees for data multicast. We choose Delaunay Triangulation (DT) as an example for mesh formation, and study various tree construction mechanisms based on the degree of embedding of the tree branches in the mesh (embedded, bypass and intermediate trees). Using simulation on Internet-like topologies, we show that SALM achieves low costs in terms of relative delay penalty and physical link stress, even for large multicast groups (in excess of a thousand end-hosts).

I. Introduction

In order to overcome current limitations in IP multicast, researchers have been focusing on enabling multicast at the application layer, the so-called application-level multicast (ALM). In some multicast applications such as stock quotes and news-on-demand, programs are to be distributed to subsets of a pool of potential end-hosts.[1] Each program has a single source which may be distributed in the network. While the lifetime of the end-hosts in the system is relatively long (in the order of hours or days) and hence the pool is rather stable, users may hop from one program to another quite frequently (e.g., in the order of once per minute). Since each program is delivered to a dynamic subset of end-hosts, building efficient multicast trees for these programs becomes an important issue.

One obvious but naive approach is to broadcast all the programs to all the end-hosts and let them select the programs. This is clearly not efficient in terms of bandwidth and message delay, especially for those unpopular programs. Maintaining a separate and distinct tree for each program using the traditional mesh-based or tree-base ALM protocols appears to be another solution. However, the mesh-based approach (such as Narada or NICE) would lead to too many meshes to be maintained in the system [1], [2]. Furthermore, as users hop from one mesh to another, mesh reformation would lead to high packet loss. Similarly, using tree-based protocols (such as YOID or HMTP) to build independent trees for each multicast group is not efficient due to high control overhead to detect and eliminate loops [3], [4].

To address the above problem, we propose a framework which builds multiple trees based on a *single* shared overlay mesh. The mesh is formed by all the end-hosts in the system and hence is independent of joining and leaving events in any groups. This relatively stable mesh is used for control messaging and for efficiently guiding the formation of overlay trees, which run on the top of the mesh for data delivery. Even though the number of overlay trees is as many as the programs, they can be efficiently constructed loop-free with the help of the mesh. Since an overlay tree serves only a subset of end-hosts in the network, we termed this framework Subset-ALM, or SALM for short. Our framework may use any existing mesh-based ALM protocol where the nodal relationship (i.e., whether a node is a child or the parent of another connected node) can be determined easily.

In this paper, we use Delaunay Triangulation (DT) as an example. We investigate the following two important issues in SALM based on a DT mesh:

- *Mesh formation and maintenance*
  The single mesh has to be formed and maintained efficiently in order to reduce network control and delivery overhead. Efficient estimation of the relative location of hosts in the Internet is hence very important. Using *Global Network Positioning (GNP)* (or many other equally good ones [5]–[8]), an efficient DT mesh with low transmission overhead can be formed.
- *Construction of overlay trees for data delivery*
  Given the mesh, we study how source-specific trees for data delivery can be efficiently constructed and maintained. We consider three ways to construct such an overlay tree: 1) Embedded tree, where the tree branches consist of the mesh edges, 2) Bypass tree, where only the group members form the tree, and the mesh edges may

This work has been supported, in part, by the Hong Kong Areas of Excellence (AoE) Scheme on Information Technology of the University Grant Council (AoE/E-01/99), and by grants of the Research Grant Council (HKUST6199/02E & HKUST6156/03E) in Hong Kong.

[1]In this paper, we refer to an "end-host" as a client machine where a user sits at. It joins and leaves programs on the user's behalf.

0-7803-8924-7/05/$20.00 (C) 2005 IEEE

be bypassed, and 3) Intermediate tree, where some nodes not belonging to the group may belong to the tree.

As compared to traditional tree-based protocols, SALM achieves much lower control overhead. This is due to the overlay mesh, which allows SALM to discover nodal relationship easily. Furthermore, the network delay between any two hosts can also be properly estimated given their relative locations in the GNP space, thus saving much overhead in delay measurement.

This paper is organized as follows. We discuss in detail how to form and maintain the mesh in Section II. In Section III, we present the construction of overlay trees on the mesh. Based on Internet-like topologies, we illustrate representative simulation results in Section IV. We conclude in Section V.

II. Mesh Formation and Maintenance

In this section, we discuss in detail how SALM builds an efficient overlay mesh on top of all end-hosts.

A. Accurate Estimation of Host Locations using GNP

In the traditional DT protocol, hosts first form a mesh based on their *geographical* locations [9]. Compass routing, a kind of local routing, is used to route a message from one point to another. The strengths of this protocol are that: (i) a host only needs to maintain the states of its immediate neighbors to construct the overlay mesh; and (ii) a host exchanges information only with its neighbors, and does not need to know the states of other non-neighbor hosts.

However, DT estimates the host locations based on their geographic coordinates. This may work well for wireless networks, but not so for the Internet where the delay between any hosts does not correlate well with their geographical locations. A better estimation of the host location is hence important.

SALM uses Global Network Positioning (GNP) to properly estimate host locations in the *Internet space* (as mentioned before, any other network positioning systems may be used).

GNP has been proposed in [5] as a way to estimate the relative location of a host in the Internet such that the difference between the locations of two hosts correlates well with the round-trip time between them.

In GNP, a number of infrastructure hosts termed as *landmarks* are used as reference points for measurement purposes. The landmarks, after measuring the round-trip time among themselves, forward the measurement results to one of the landmarks, which uses the results to compute the landmark locations in the GNP (or *Internet*) space by minimizing an objective function. The locations are then disseminated back to the respective landmarks.

Note that the landmarks of GNP in SALM are not likely to be flooded with ping requests. According to [10], most of the Internet routes are stable for a long time (about 80% routes longer than a day). Therefore, a host does not needs to probe landmarks frequently to estimate its location. Furthermore, since computation of host coordinate is done locally at the end-hosts, a landmark does not need to store any information or perform computation upon the arrival of a new host.

B. Join Mechanism

A joining host, after obtaining its location (according to the GNP mechanism given in Section II-A), sends a *MeshJoin* message with its location to any host in the system (can be the multicast source). *MeshJoin* is then sent back to the joining host along the DT mesh according to compass routing. Since the joining host is not a member of the mesh yet, it can be considered as a partitioned mesh of a single host. As the *MeshJoin* message is forwarded, it triggers the partition recovery mechanism at a particular host on the periphery of the DT mesh to connect the joining host to the mesh.

In Figure 1, we illustrate the mechanism involved in member joining. Suppose that u is the joining host. The following are the steps on how u joins the mesh:

1) u first retrieves the list of landmarks by querying a host b with a *GetLandmark* message (Figure 1(a)).
2) Then u measures the round-trip time to those landmarks (e.g., through ICMP messages) to estimate its location (Figure 1(b)).
3) After that, u sends a *MeshJoin* message to b again (Figure 1(c)).
4) The message is then forwarded from b to c with compass routing (Figure 1(d)).
5) Since u falls inside $\triangle acd$, c knows that u is on another partitioned mesh, therefore c adds u into its neighbor list N_c to recover the partition.
   Note that the minimum internal angle of $\triangle auc$ and $\triangle abc$ is less than that of $\triangle buc$ and $\triangle abu$. Therefore the connection from c to a violates the DT property, and c will remove a from N_c (Figure 1(e)).
6) c then broadcasts its neighborhood information to its neighbors with *HelloNeighbor* messages. Upon receiving them, b and d discover u and adds u into their neighbor list. In the meantime, u also discovers b and d and adds them into its neighbor list (Figure 1(f)).
7) Suppose that b is the next host to broadcast its neighborhood information through *HelloNeighbor* messages. Upon receiving the message, a discovers u, and adds u into its neighbor list. Afterwards, a also notices that the connection from a to c violates the DT property, and hence removes c from its neighbor list. The resultant overlay mesh thus satisfies the DT property after the joining of u (Figure 1(g)).

III. Construction of Data Delivery Trees

In this section, we present three algorithms to construct overlay trees for data delivery on top of the SALM mesh. We then describe how SALM maintains its loop-free property.

A. Embedded, Bypass and Intermediate Trees

We study three algorithms to build trees in SALM. The first is so-called *Embedded Tree*, which builds an overlay tree such that all edges are part of the overlay mesh. Clearly, in forming the tree, non-member nodes may be included. The second one builds an overlay tree such that it covers only the group members without having to use the mesh edges. Since the

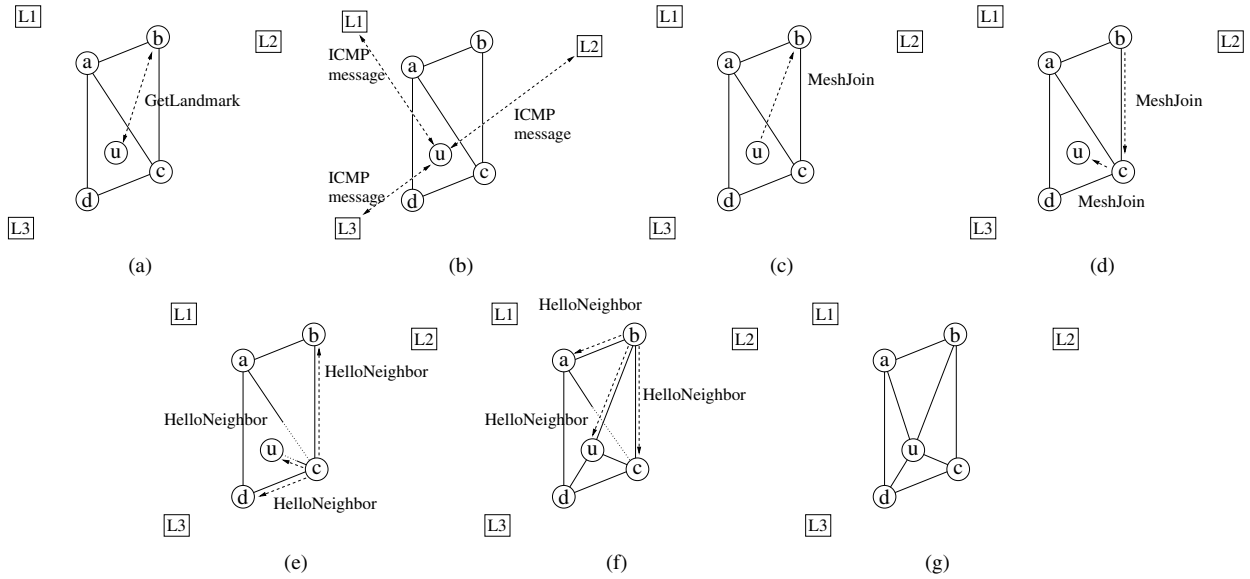


Fig. 1. Example scenario of host joining.

tree *bypasses* the non-member nodes and mesh edges to form direct connections with only the group members, we term such a tree *Bypass Tree*. The third one is termed *Intermediate Tree*, which lies between embedded and bypass tree where some non-member nodes may be included in the tree. We define any non-leaf node in the overlay tree as *forwarder*, which needs to forward data messages to its children. We elaborate the mechanism in detail in the following.

- *Embedded Tree*
  To form an embedded tree, a joining host first sends a *TreeJoin* message to the source along the DT mesh using compass routing. All hosts along the path become forwarders no matter whether they are interested in the program or not. We explain how the *TreeJoin* message is handled in a host in the mesh: A host first adds the joining host into its children table for the specified program. Then it checks whether it is already a forwarder of the program as specified in the *TreeJoin* message. If so, it suppresses the forwarding of the message. Otherwise, it turns itself into a forwarder and relays the *TreeJoin* message to the program source.
- *Bypass Tree*
  Recall that all forwarders in a bypass tree are interested in the data from the program source. A joining host sends a *TreeJoin* message upstream to the source using compass routing. A non-member host receiving the message simply relays the message to the next hop (using compass routing) without turning itself into a forwarder. On the other hand, if the host is a member of the program, it adopts the joining host as its child by adding the joining host into its children table and suppresses the forwarding of the message.
- *Intermediate Tree*
  Observe that embedded tree requires the participation of non-member hosts, i.e., a host may need to serve other hosts of different multicast groups. Therefore, as compared to bypass tree, it demands more local resources. Moreover, it consumes more network resources and has longer delay, especially for sparse groups. On the other hand, a node in a bypass tree may suffer from high nodal degree and hence loading (a star-like topology rooted at the source for sparse group).
  Therefore we propose intermediate tree which builds a tree between embedded tree and bypass tree by including in the tree those non-member nodes receiving more than a certain number of join messages. This is done so in order to keep both nodal degree and delay low. More specifically, a host handles the request according to the bypass tree if the number of request received is less than a certain threshold; otherwise, the host forwards the request according to embedded tree.

B. Loop-Free Maintenance

Traditionally, maintaining a loop-free overlay tree can be quite costly. In the event of joining and leaving, a loop is formed if a host accepts a connection request from its ancestor. One approach to avoid looping is that, before adopting a node as its child, a host sends control messages to trace the path from itself to the root. If the node appears on the trace path, the host rejects it. We show an example in Figure 2, where arrows represent the parent-child relationship. Suppose a sends a connection request to e to be its child. Before e accepts the request, it traces the path to the root (i.e., $e \rightarrow d \rightarrow a \rightarrow s$). Since a is on the path, e rejects the connection request. Clearly, though this algorithm works, it introduces much control message overhead.

SALM does not rely on such a trace-back to avoid looping, because the tree formed by compass routing is inherently loop-free all the time. This is due to the fact that compass routing

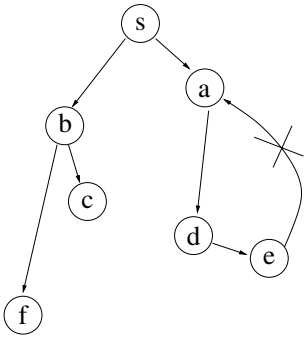


Fig. 2. A loop in a data delivery tree.

in DT is a greedy algorithm — the GNP distance to a host from the source strictly decreases along the path [11]. In other words, in a data delivery tree, the GNP distance from a host to the source is always shorter than that from its offspring. This important property leads to the loop-free characteristic in SALM.

The loop-free proof is by contradiction as follows. Suppose a loop exists in the delivery tree with source (root) s, say, going from a host i to some node j before going back to i again. For the path i to j, the property above states that the GNP distances $si < sj$ (since i is an intermediate node from s to j). However, for the path j to i, the property above states that $sj < si$ (j is an intermediate node from s to i). This is clearly a contradiction, meaning that SALM is loop-free.

Besides the loop-free characteristics, SALM tree construction also achieves low measurement overhead. Most traditional ALM protocols rely on sending frequent probes to measure the network delay between hosts, so that the overlay trees can be built with short edges. In SALM, the distance between two hosts can be estimated based on the difference between their GNP coordinates. This greatly reduces the overhead for tree optimization without compromising much of its performance.

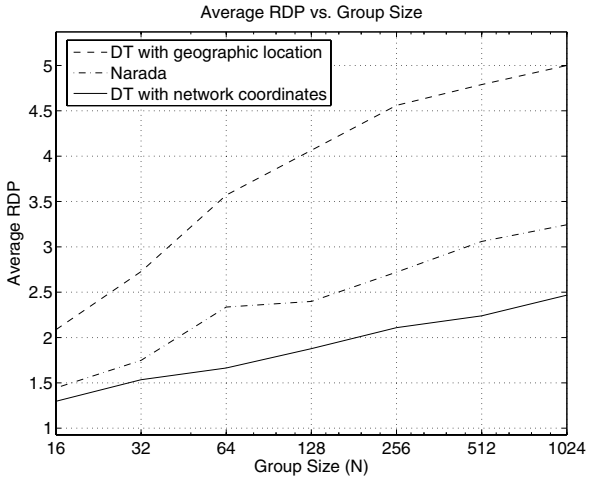
IV. Illustrative Numerical Results

A. Simulation Setup

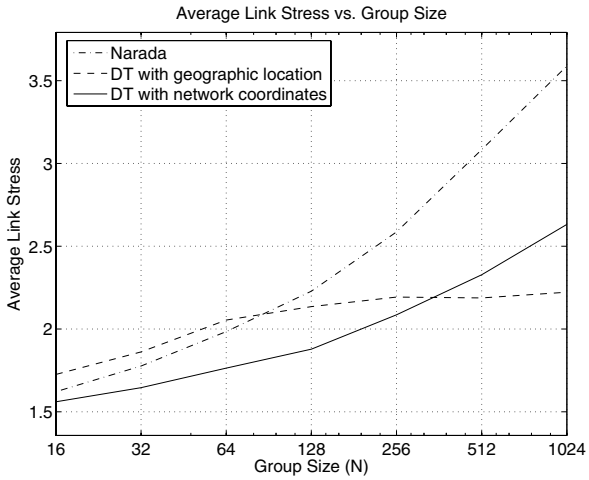
We generate *Transit Stub* topologies with GT-ITM [12]. The parameters used for topology generation are according to the study of the traditional DT protocol in [9]. The generated topologies are a two-layer hierarchy of transit networks (with four transit domains, each with 16 randomly-distributed routers on a 1024×1024 grid) and stub networks (with 64 domains, each with 15 randomly-distributed routers on a 32×32 grid). A host is connected to a stub router via a LAN (of 4×4 grid). The delay of LAN links is 1ms while the delay of core links is computed by the topology generator.

For GNP, we select a number (20) of landmarks based on N-cluster-median criterion as given in [5]. For each DT mesh, we randomly select a host as the source which multicasts packets along the DT mesh to all members with compass routing. We are interested in the following performance metrics:

- Relative delay penalty (RDP), defined as the ratio between overlay delay to underlay delay of a host from the source;
- Physical link stress, defined as the number of identical packets passing through a link, given the packet passes through the link.



a) Average RDP.



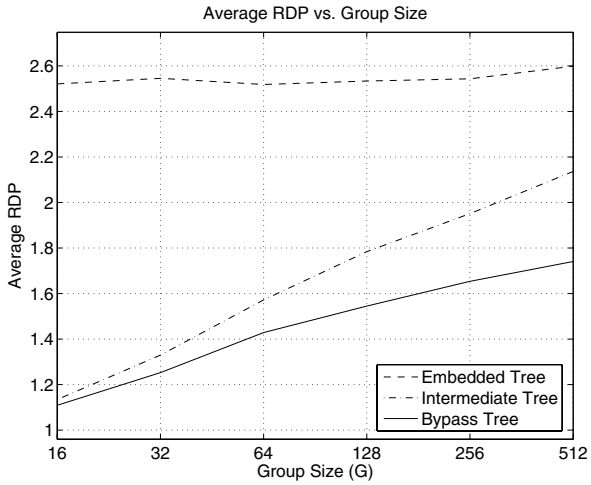
b) Average link stress.

Fig. 3. Performance comparison of DT mesh formation using GNP, geographic location, and Narada.

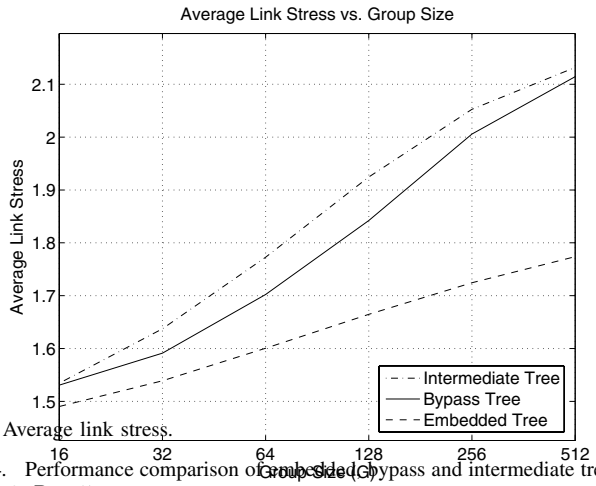
Unless otherwise stated, the parameters we use are $N = 1024$ (a total of 1024 end-hosts in the system), $G = 128$ (128 of them belong to the same multicast group), $K = 8$ (the maximum fanout of a host is 8), and $R = 8$ (in intermediate tree, a non-member host handles requests according to the bypass tree if the request number is less than 8; otherwise, it acts according to the embedded tree).

B. SALM Performance

We first compare SALM performance with a traditional scheme, Narada. Because traditional schemes have not considered subset multicast, for fair comparison, we have set $N = G$. In this case, SALM is the same as DT. We further compare the cases with network coordinates (GNP) and without (i.e., using geographic location). Figure 3(a) shows the average RDP



a) Average RDP.



b) Average link stress.

Fig. 4. Performance comparison of embedded, bypass and intermediate trees ($K = 8, R = 8$).

versus group size ($N = G$). In general RDP increases with the group size. SALM with GNP performs the best. This is because GNP coordinates correlate well with the relative location of hosts in the Internet. The SALM mesh formed using GNP is hence better than the one based on geographic locations. As the group size increases, the performance margin is more remarkable. For a medium group size ($\approx$ 128–256 members), RDP is markedly lower than Narada. Regarding link stress (Figure 3(b)), SALM with network coordinates performs better than Narada. It also performs better than using geographic location for sparse network. For dense group, using geographic location may perform better. This is because the higher the member density, the higher is the probability that the SALM mesh with GNP have "small angles.". This means that end-to-end connections are more likely to pass through the same underlay link, leading to high stress.

We compare the performance of embedded, bypass, and intermediate trees versus group size in Figure 4. We show RDP in Figure 4(a). The RDP of bypass tree is significantly lower than that of embedded tree, while the intermediate tree lies between them. For small groups, bypass tree skips nearly all mesh edges; therefore, its RDP is close to one. The RDP for embedded tree is rather independent to the group size as it forms trees covering the mesh edges.

Figure 4(b) compares link stress for the three schemes. The embedded tree has the lowest stress, because the forwarding load is distributed among all the nodes in the network. Bypass and embedded trees suffer higher stress due to their higher nodal fanout (node stress), the number of identical packets replicated by a given host. The intermediate tree has slightly higher stress than bypass tree, mainly due to high stress in some nodes.

In summary, the figures suggest that, overall, bypass tree works well; it achieves low RDP and intermediate stress performance. For intermediate tree, its RDP lies between bypass and embedded trees, especially for small to medium group size. (It is difficult to design an intermediate tree which lies between bypass and embedded trees for all metrics due to subtle trade-offs among the metrics.)

V. Conclusion

In application-layer multicast (ALM), users may hop from one group to another quite frequently. In this paper, we propose a novel ALM framework for dynamic groups termed subset-ALM (SALM). SALM supports multiple multicast groups and efficiently distributes data to dynamic subsets of a pool of end-hosts. It first builds a shared overlay mesh for all the end-hosts in the system. This rather stable mesh is then used to guide the construction of overlay trees for data delivery to each group. We study three ways of constructing the trees. Our simulation results on Internet-like topologies show that SALM achieves low RDP and link stress, even for large groups (more than hundreds of hosts).

References

[1] Y. hua Chu and S. G. R. S. S. H. Zhang, "A case for end system multicast," *IEEE JSAC*, vol. 20, pp. 1456–1471, Oct. 2002.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. SIGCOMM'02*, pp. 205–217, Aug. 2002.

[3] P. Francis, P. Radoslavo, R. Lindell, and R. Govindan, "Your own internet distribution YOID." http://www.isi.edu/div7/yoid/.

[4] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," in *Proc. INFOCOM'02*, 2002.

[5] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. INFOCOM'02*, June 2002.

[6] L. Tang and M. Crovella, "Virtual landmarks for the Internet," in *Proc. IMC'03*, Oct. 2003.

[7] H. Lim, J. Hou, and C.-H. Choi, "Constructing Internet coordinate system based on delay measurement," in *Proc. IMC'03*, Oct. 2003.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM'04*, Aug. 2004.

[9] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with delaunay triangulation overlays," *IEEE JSAC*, vol. 20, pp. 1472–1488, Oct. 2002.

[10] Y. Zhand, V. Paxson, and S. Shenker, "The stationarity of internet path properties: routing, loss and throughput," Tech. Rep., ACIRI, May 2000.

[11] E. Kranakis, H. Singh, and J. Urrutia, "Compass routing on geometric networks," in *Proc. CCCG'99*, pp. 51–54, Aug. 1999.

[12] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. INFOCOM 1996*, 1996.