Scalable Peer Finding on the Internet

Suman Banerjee, Christopher Kommareddy, Bobby Bhattacharjee
Department of Computer Science, University of Maryland, College Park

Abstract—**We consider the problem of finding nearby application peers over the Internet. We define a new peer-finding scheme (called Tiers) that scales to large application peer groups. Tierscreates a hierarchy of the peers, which allows an efficient and scalable solution to this problem. The scheme can be implemented entirely in the application-layer and does not require the deployment of either any additional measurement services, or well-known reference landmarks in the network.**

We present detailed evaluation of Tiers and compare it to one previously proposed scheme called Beaconing. Through analysis and detailed simulations on 10,000 node Internet-like topologies we show that Tiers achieves comparable or better performance with a significant reduction in control overheads for groups of size 32 or more.

I. Introduction

Consider a distributed peer-to-peer application, such as Gnutella. When a new member joins the application, it often has to find another peer that is already part of the application. Usually, the goal is to find another application peer that is "near" the new host. We refer to this problem as the peer-finding problem. Efficiently locating nearby peers is an important problem for many applications, including the emerging peer-to-peer applications. For applications like Gnutella, finding the nearest peer can reduce network load for queries and responses. Distributed data storage and lookup services like Chord [17] and CAN [14] is more efficient when the neighbors on the overlay are near to each other on the underlying topology. Nearest-peer finding techniques can also be used to efficiently construct overlay networks [1], [18], [2], [7], [5], [19] where proximity between overlay peers is desirable, to locate nearby mirrors for file transfers [6], to locate nearby sources in content distribution networks [1], to naturally implement application-layer anycasting services [3], [11].

In our prior work, we had introduced a peer-finding technique called Beaconing [12], that efficiently, quickly and accurately finds nearby peers on the Internet, but does not scale to large application groups. In this paper, we present a new peer-finding technique (called Tiers [2]) that is specifically designed to scale to *large* application groups. Like Beaconing, Tiers is implemented on an unicast-only network. There are specific challenges that need to be addressed for such unicast-only solutions, which are outlined in [12].

A. Existing Approaches

We can classify peer-finding techniques based on the amount of infrastructure support that they require. The peer-finding problem can be effectively solved by using anycast services [13], [3], [11]. All application peers can join an anycast group, and the closest peer is then found by simply sending a message to the group. However, it would require global anycast support from all participating domains. An Internet-measurement infrastructure such as IDMaps [8] can be used to efficiently solve this problem. However, it requires Internet-wide deployment of the measurement entities.

Techniques that require a very limited infrastructure support include a triangulation method, due to Hotz [10], and its weighted variant [9], a "distributed binning" technique [15] and Beaconing [12]. These techniques use a small set of measurement reference points in the network called landmarks [15] or beacons [12]. Distances between each application peer and these beacons are measured, and are processed to obtain the nearest peer. Therefore, the control overheads (e.g. for distance measurements) at each of these beacons are $O(N)$, where N is the size of the group.

B. Tiers Approach

In the Tiers technique, proposed in this paper, we create a hierarchy of the application peers. The hierarchy is based on topological clustering of these peers, where nearby peers are grouped into the same cluster. The querying member (termed query-host) finds its closest peer by successively refining its search in a top-down manner over this hierarchy.

The Tiers technique has benefits over previously known techniques in two significant ways:

- *No infrastructural support required*: All the prior proposed schemes rely on the existence of some infrastructure support. Schemes based on GIA and IDMaps would require a widespread deployment of these mechanisms on the Internet. Schemes based on Hotz triangulation, Distributed Binning and Beaconing requires the existence of special landmark entities (referred to as landmarks or beacons) which serve as the reference point for different proximity tests.

In contrast, the Tiers scheme requires no such support. In this scheme, each application peer dynamically discovers a few other application peers, and is required to make distance measurements to a subset of them.

- *Scalability*: Because of its use of an appropriate peer hierarchy, the Tiers scheme scales well with increase in the application group size. More specifically, the worst case storage and communication overhead at any entity (application peers or query-host) in this technique is bounded by $O(\log N)$, while the overheads at an average entity is a constant. The query latency is also bounded by $O(\log N)$. In this paper, we study in detail the tradeoffs of the marginal increase in query latency to the significant reduction in control overheads.

II. Tiers : Scalable Peer Finding

The Tiers peer-finding technique arranges the set of application peers into a hierarchy; the basic operation of the protocol is to create and maintain the hierarchy. We use the same hierarchy construction scheme that was used to define a scalable application layer multicast protocol [2]. Use of this hierarchy enables scalability, since most peers are at the bottom of the hierarchy

[1] See http://www.stardust.com/cdn

[2] Our work is not related to the Tiers topology generator by M. Doar.

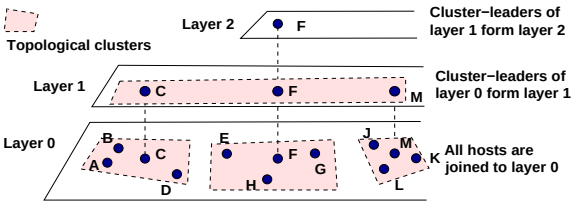

Fig. 1. Hierarchical arrangement of peers in Tiers. The layers are logical entities overlaid on the same underlying physical network.

and only maintain state about a constant number of other peers. The peers at the very top of the hierarchy maintain (soft) state about $O(\log N)$ other peers. Logically, each peer keeps detailed state about other peers that are *near* in the hierarchy, and only has limited knowledge about other peers in the group. The hierarchical structure is also important for localizing the effect of peer failures.

While constructing the Tiers hierarchy, peers that are "close" with respect to the distance metric are mapped to the same part of the hierarchy. In this paper, we use end-to-end latency as the distance metric between hosts. We leverage this topological arrangement in efficiently identifying the closest peer, with a small number of probes. The closest peer-finding operation proceeds top-down on the hierarchy thus successively refining the search at each step, till the appropriate peer is identified.

In the rest of this section, we briefly describe how the Tiers hierarchy is defined. Details and specifics of the hierarchy construction mechanism can be found in [2].

A. Hierarchical Arrangement of Application Peers

The Tiers hierarchy is created by assigning peers to different levels (or layers) as illustrated in Figure 1. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denoted by L_0). Hosts in each layer are partitioned into a set of clusters. Each cluster is of size between k and $2k-1$, where k is a constant, and consists of a set of hosts that are close to each other. Further, each cluster has a cluster leader. The protocol distributedly chooses the (graph-theoretic) center of the cluster to be its leader, i.e. given a set of hosts in a cluster, the cluster leader has the minimum maximum distance to all other hosts in the cluster. The cluster leader, is therefore, an approximation of the location of all the cluster peers.

Hosts are mapped to layers using the following scheme: All hosts are part of the lowest layer, L_0. The clustering protocol at L_0 partitions these hosts into a set of clusters. The cluster leaders of all the clusters in layer L_i join layer L_{i+1}. This is shown with an example in Figure 1, using $k = 3$. The layer L_0 clusters are [ABCD], [EFGH] and [JKLM][3]. In this example, we assume that C, F and M are the centers of their respective clusters of their L_0 clusters, and are chosen to be the leaders. They form layer L_1 and are clustered to create the single cluster, [CFM], in layer L_1. F is the center of this cluster, and hence its leader. Therefore F belongs to layer L_2 as well.

The Tiers clusters and layers are created using a distributed algorithm as described in [2].

[3]We denote a cluster comprising of hosts $X, Y, Z, \ldots$ by $[XYZ\ldots]$.

B. Finding the closest peer

The closest peer finding operation proceeds top down on the peer hierarchy. We assume the existence of a special host that the query-hosts know of a-priori through out-of-band mechanisms. We call this peer the Boot Strap Host (BSH) [4]. Each query-host initiates the query process by contacting the BSH. For ease of exposition, we assume that the BSH is the leader of the single cluster in the highest layer of the hierarchy. bypassed on the data path. (Alternatively is is possible that the BSH is only aware of the leader of the highest layer cluster, and therefore, not itself be part of the hierarchy. We do not belabor this complexity further.)

We illustrate the query procedure using the example shown in Figure 2. In the figure, we the application group has already been arranged into four L_0 clusters (marked by dotted lines). Hosts C_0, B_0, B_1 and B_2 are the leaders of these respective clusters. They together form a single cluster in layer L_1. The leader of this L_1 cluster is C_0, and is the only host in layer L_2.

Assume that host A_1 wants to find its closest peer in this group. First, it contacts the BSH with its query (Panel 0). The BSH responds with the hosts that are present in the highest layer of the hierarchy. The query-host then contacts all peers in the highest layer (Panel 1) to identify the peer closest to itself. In the example, the highest layer L_2 has just one peer , C_0, which by default is the closest peer to A_1 amongst layer L_2 peers. Host C_0 informs A_1 of the three other peers (B_0, B_1 and B_2) in its L_1 cluster. A_1 then contacts each of these peers with the query to identify the closest peer among them (Panel 2), and iteratively uses this procedure to find the closest L_0 cluster (whose leader happens to be B_2). Finally, it queries each of these L_0 cluster peer (Panel 3), and is thus able to select the closest of these peers (i.e. A_0) as its closest peer in the application group.

It is important to note that any host, H, which belongs to any layer L_i is the center of its L_{i-1} cluster, and recursively, is an approximation of the center among all peers in all L_0 clusters that are below this part of the layered hierarchy. Hence, querying each layer in succession from the top of the hierarchy to layer L_0 results in a progressive refinement in finding the closest peer. The outline of this operation are presented in pseudocode as Procedure *FindClosest* in Figure 3.

Occasionally it might happen that the cluster membership information at the leader is stale (e.g. all members of the cluster suddenly left the application peer group). The cluster leader detects this situation when it does not receive appropriate *heartbeat* messages from its members. In such cases, the query-host is unable to elicit responses from any of these members returned by the cluster leader. In such cases, the query-host re-initiates the query from the previous layer cluster leader, which had responded to it. In the worst case, the query is re-initiated from the BSH.

C. Invariants

The following properties hold for the distribution of hosts in the different layers:

- A host belongs to only a single cluster at any layer.
- If a host is present in some cluster in layer L_i, it must occur in one cluster in each of the layers, $L_0, \ldots, L_{i-1}$. In fact, it is the

[4]It is same as the host known as the Rendezvous Point in [2].

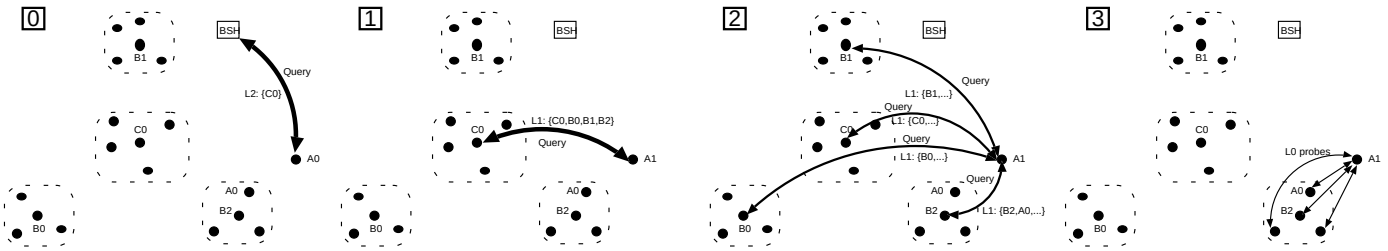

Fig. 2. Query-host A_1 finds its closest peer (A_0).

Procedure : *FindClosest*(h)
Cl$_j$ $\leftarrow$ *Query*($BSH, -$)
$while\ (j \geq 0)$
  Find y s.t. d$ist(h, y) \leq$ d$ist(h, x), x, y \in$ *Cl*$_j$
  $if\ \ (j = 0)$
    $return\ y$
  $endif$
  Cl$_{j-1}(y)$ $\leftarrow$ *Query*($y, j - 1$)
  Decrement j, *Cl*$_j$ $\leftarrow$ *Cl*$_{j-1}(y)$
$endwhile$

Fig. 3. Basic query operation for peer h. *Query*($y, j - 1$) seeks the membership information of *Cl*$_{j-1}(y)$ from peer y. *Query*($BSH, -$) seeks the membership information of the topmost layer of the hierarchy, from the BSH.

cluster-leader in each of these lower layers.

- If a host is not present in layer, L_i, it cannot be present in any layer L_j, where $j > i$.
- Each cluster has its size bounded between k and $2k - 1$. The leader is the graph-theoretic center of the cluster.
- There are at most $\log_k N$ layers, and the highest layer has only a single peer.

All the good properties of this scheme (as analyzed next) hold as long as the hierarchy is maintained. Thus, the objective of the distributed Tiers protocol, described in [2], is to scalably maintain the host hierarchy as new peers join and existing peers depart.

D. Analysis

We analyze the efficiency of this peer-finding scheme by evaluating the storage requirements for peer state, communication overheads to exchange control messages for maintaining the hierarchy and the latency incurred by the query-host in identifying the closest peer.

Each cluster in the hierarchy has between k and $2k - 1$ peer. Then, a host that belongs only to layer L_0 maintains state for only $O(k)$ other hosts and incurs an equivalent communication overhead for exchange of control messages. In general, a host that belongs to layer L_i and no other higher layer, maintains state for $O(k)$ other hosts in each of the layers $L_0, \ldots, L_i$. Therefore, the control overhead for this peer is $O(k.i)$. Hence, the cluster-leader of the highest layer cluster (Host C_0 in Figure 2), maintains state for a total of $O(k \log N)$ neighbors. This is also the worst case control overhead at a peer.

It follows using *amortized cost* analysis that the control overhead at an average peer is a constant. The number of peers that occur in layer L_i and no other higher layer is bounded by $O(N/k^i)$. Therefore, the amortized control overhead at an average peer is

$$\leq \frac{1}{N} \sum_{i=0}^{\log N} \frac{N}{k^i} k.i = O(k) + O(\frac{\log N}{N}) + O(\frac{1}{N}) \rightarrow O(k)$$

with asymptotically increasing N. Thus, the control overhead is $O(k)$ for the average peer, and $O(k \log N)$ in the worst case.

The query process incurs a message overhead of $O(k \log N)$ query-response pairs. The query-latency depends on the delays incurred in these exchanges, which is typically about $O(\log N)$ round-trip times.

III. Simulation Experiments

We have analyzed the performance of Tiers using detailed simulations on very large network topologies. The topologies were generated using the Transit-Stub graph model, using the GT-ITM topology generator [4]. All topologies in these simulations had $10,000$ routers with an average node degree between 3 and 4. Application peers were attached to a set of routers, chosen uniformly at random. The number of such peers in the multicast group were varied between 8 and 512 for different experiments. We also placed 500 query-hosts, again distributed uniformly at random, on the topology, that sought to find the closest peer using the proposed Tiers approach and the beaconing approach [12] for comparison. For the beaconing scheme, we use a set of seven beacon hosts located on the topology as was shown to be appropriate in [12]. We ran between 10 and 20 instances for each experiment, to get a tight bound on the variations in the results.

We observe three different metrics in this study:

- *Accuracy:* of the different schemes in finding the closest peer.
- *Query latency:* measured from the instant the query for the nearest peer is initiated by the query-host upto the time when this query is resolved.
- *Control overheads:* of the different schemes for the application peers and other entities.

We studied two different aspects of the protocols in this paper — the behavior of the different schemes as we varied the number of application peers, and the effect of membership changes to the group of application peers.

For both Beaconing and Tiers we choose the same periodic rate (of once every 5 seconds) with which the soft states are re-

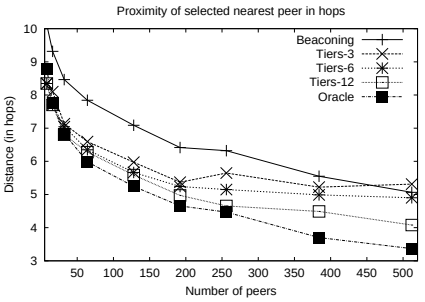

Fig. 4. Accuracy of the queries. 'Oracle' indicates the actual closest peer. (Varying peer group sizes).

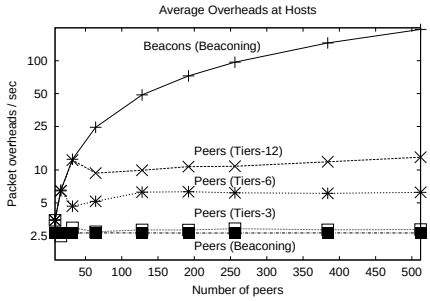

Fig. 5. Average control overheads at the end-hosts.

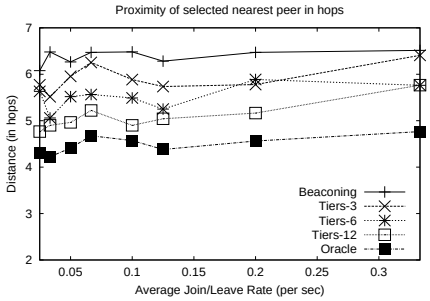

Fig. 6. Accuracy of queries. 'Oracle' indicates the actual closest peer. (Varying join/leave rates).

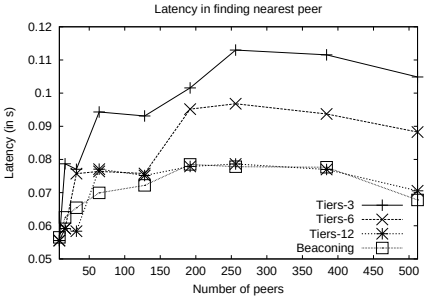

Fig. 7. Query latency (Varying peer group sizes).

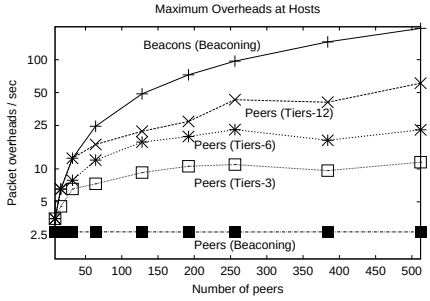

Fig. 8. Worst case control overheads at the end-hosts.

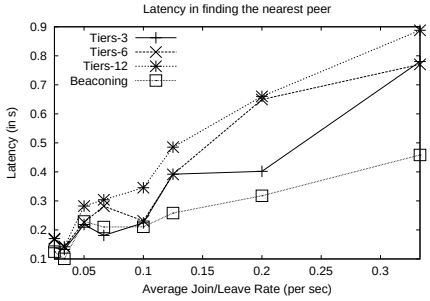

Fig. 9. Query latency (Varying join/leave rates).

freshed. For the Tiers scheme, we also varied the cluster size parameter, k to study its effect on the different metrics. In the different plots Tiers-3, Tiers-6 and Tiers-12 represent the data for the Tiers scheme with the cluster size parameter, k, set to 3, 6, and 12 respectively. In this implementation, the cluster size upper bound was relaxed to $3k - 1$ to avoid a cluster split and a merge operation to occur in quick succession that may occur otherwise in some special cases. This is explained in [2]. This does not change the analysis (in Section II) or the nature of the results.

Broadly, our findings can be summarized as follows: *The accuracy of the query and the latency incurred to satisfy the query are similar for both Tiers and Beaconing. However, due to its hierarchical structure, the worst case overheads at hosts in Tiers are significantly lower.*

Application Peer Group Size

We varied the application group between 8 and 512 to study the effects of the application peer group size on the different metrics. In Figure 4 we plot the accuracy of the different schemes (Tiers and Beaconing) in finding the closest peer. In the plot, 'Oracle' indicates the distance of the query-host to its actual closest peer on the topology. Tiers-12 performs the best among all the schemes. In particular, for the Tiers schemes, larger the cluster size, the more accurate is the result of the query. Beaconing performs somewhat less accurately than Tiers, however the difference between the schemes is relatively low.

In Figure 7, we plot the latency of the queries for the same set of topologies. Beaconing and Tiers-12 has very similar latencies. In particular, with asymptotically increasing size, the query latency for Beaconing does not depend on the peer group size and therefore is expected to be the lowest. For the Tiers schemes, the query latency increases very slowly (logarithmically) with the increase in peer group size. In Figure 7, the latency for the Tiers schemes do not increase smoothly for a given cluster parameter, k. This is because the query latency increase only when the number of layers increase, i.e. when the peer group size approximately increases by a factor of k, as can be observed in the plots. For peer group sizes that have the same number of layers, the latency actually decreases with increase in group size. This is because, as the topology gets more and more populated by application peers, the nearest peers are effectively closer to the query-host.

Finally, in Figures 5 and 8 we plot the average and the maximum control traffic overheads at the hosts. Note that Y-axis in the figures are plotted in the log scale. For all the group sizes simulated, the overheads at the average application peers for Beaconing and Tiers-3 are very similar (about 2.6 packets/second). The overheads for Tiers-6 and Tiers-12 are correspondingly higher (2.9 and 13.0 packets/second respectively for groups of size 512). In contrast, the overheads at the beacons (in Beaconing) are about 193.5 packets/second for the same size group. The maximum packet overhead at any host for Tiers-6 is significantly lower than than Beaconing for groups of size 32 or more, is about an order of magnitude lower (22.8 packets/second) for the peer groups of size 512. The overheads at the beacons increase linearly with the peer group size, where as the worst case overheads for Tiers increase logarithmically. Therefore, the Beaconing scheme is efficient and fast for small groups, but does not scale with increasing group sizes.

In Tiers, as the cluster size parameter, k is increased the con-

trol overheads at the hosts also increase. However, the query accuracy and query latency correspondingly decrease. Therefore, this parameter can be appropriately chosen to trade-off between the protocol performance and control overheads.

Changing Group Membership

Next, we studied the effects of dynamic changes to the group membership for the different schemes. In this experiment, a set of 256 application peers initially joined the group over a 200 second period. Subsequently, new application peers joined and existing peers left the group uniformly at random at a specified rate. We varied this average join/leave rate from moderately changing groups (i.e. 1 change per 40 seconds, a rate of 0.025/s) to very rapidly changing groups (i.e. 1 change every 3 seconds, a rate of 0.33/s). In Figure 6, we plot the accuracy of the results for this experiment. As can be observed, the accuracy of the result is not significantly impacted for these change rates.

The query latency, however, increases significantly for the high change rate scenarios (Figure 9). Note that for the most dynamic scenarios, the join/leave rate is faster than the periodic refresh rates (of one every 5 seconds) used for the schemes. For the Beaconing scheme, the responses from the beacons might include peers that have already left the group, leading to re-initiating the query. The high join leave rate has a greater impact on the Tiers scheme, because in this scenario, the membership of clusters change frequently. The cluster leaders have stale information about the cluster members (some of them might have already left). This also leads to occasional re-queries at each layer in the hierarchy. This causes the corresponding increase in the query latency.

IV. Related Work

Guyton et. al. [9] present a taxonomy for locating peers on the Internet. They classify existing techniques into reactive gathering and proactive gathering categories. Expanding ring searches are classified under the reactive category. Our work can be classified under probing-based schemes along with the triangulation-based approaches due to Hotz [10] and the weighted variant [9]. The beaconing technique [12] also uses similar distance-estimation probes to find the nearest peer. However, all these other schemes require incur significantly higher traffic overhead in comparison to the Tiers approach. A completely different approach to finding the nearest peer is to use passive measurements, as described in [16], and is particularly useful if the nearest peer in the group remains relatively static.

We additionally classify peer finding techniques based on the amount of infrastructure support necessary. Anycasting [13], [11], [3] can be used o solve the nearest peer problem by grouping all peers in the same anycast group but requires universal deployment of this service. IDMaps is a global distance measurement infrastructure that needs deployment of "tracers" or measurement servers in the network. For N tracers and M peers This technique incurs $O(N^2 + M)$ overheads. Techniques like Distributed Binning [15] and Beaconing [12] use limited infrastructure support — a set of well-known landmark entities that are distributed in the network, with respect to which all distance measurements are made. In contrast, the Tiers approach uses a no infrastructure support except for the BSH — a single bootstrapping host which is necessary for all schemes.

V. Conclusions

In this paper, we have presented a new protocol for scalable peer finding on the Internet. Through detailed simulations we show that the protocol achieves similar performance while significantly lower overheads on groups larger than 32. The protocol is based on a hierarchical clustering of the peers. This technique has a wider applicability than this peer finding application.

While in this paper, we describe the protocol to find the closest peer with respect to the hop-count metric, it is applicable to other metrics. For example, by performing the hierarchical clustering based on the access bandwidths, it is easy to see that this protocol is able to find peers with similar bandwidths. Such bandwidth-based peer finding proved to be useful in the preference clustering approach for multicast data delivery to a group, where members clustered into sub-groups based on their bandwidths, and an appropriate data rate is sent to these sub-groups that best meets their capabilities.

References

[1] D.G. Andersen, H. Balakrishnan, M. Frans Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, October 2001.

[2] S. Banerjee, S. Parthasarathy, and B. Bhattacharjee. A protocol for scalable application layer multicast. In *CS-TR-4278, Department of Computer Science, University of Maryland College Park, USA*, July 2001.

[3] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proceedings of INFOCOM*, 1997.

[4] K. Calvert, E. Zegura, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom*, 1996.

[5] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.

[6] Z. Fei, E. Bhattacharjee, S. Zegura, and M. Ammar. Finding the best server within the application-layer anycasting architecture. In *Proceedings of INFOCOM*, 1998.

[7] P. Francis. Yoid: Extending the Multicast Internet Architecture, 1999. White paper http://www.aciri.org/yoid/.

[8] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An architecture for a global internet host distance estimation service. In *Proceedings of INFOCOM*, March 1999.

[9] J. Guyton and M.F. Schwartz. Locating nearby copies of replicated internet servers. In *Proceedings of SIGCOMM*, 1995.

[10] S. Hotz. Routing information organization to support scalable interdomain routing with heterogeneous path requirements. In *PhD thesis, University of Southern California*, 1996.

[11] D. Katabi and J. Wroclawski. A framework for scalable global ip-anycast (GIA). In *Proceedings of ACM SIGCOMM*, August 2000.

[12] C. Kommareddy, N. Shankar, and B. Bhattacharjee. Finding close friends on the Internet. In *Proceedings of ICNP*, November 2001.

[13] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. In *Request for Comments 1546, IETF*, November 1993.

[14] S. Ratnaswamy, P. Francis, M. Handley, K. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, August 2001.

[15] S. Ratnaswamy, M. Handley, K. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of INFOCOM*, June 2002.

[16] M. Stemm, S. Seshan, and R. Katz. A network measurement architecture for adaptive applications. In *Proceedings of INFOCOM*, March 2000.

[17] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*, August 2001.

[18] J. Touch and S. Hotz. The x-bone. In *Proceedings of 3rd Global Internet Mini-Conference at Globecom '98*, November 1998.

[19] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area dat a dissemination. In *Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.