Efficient Peer Location on the Internet

Suman Banerjee [a,*], Chris Kommareddy, Bobby Bhattacharjee [b]

[a] *Dept. of Computer Sciences, University of Wisconsin, Madison, WI 53706, USA*

[b] *Dept. of Computer Science, University of Maryland, College Park, MD 20742, USA*

Abstract

We consider the problem of locating nearby application peers over the Internet. We define a new peer-location scheme (called Tiers), that scales to large application peer groups. Tiers creates a hierarchy of peers, and provides an efficient and scalable solution to the peer-location problem. Tiers can be implemented entirely in the application-layer and does not require the deployment of either any additional measurement services, or well-known reference points in the network.

We have evaluated the performance of Tiers through detailed experiments. Our results show that Tiers is able to locate the nearest peers quickly ($\ll$ 1 second) and accurately on wide-area Internet-like topologies. We have also compared the performance of Tiers with two other schemes, Beaconing and Distributed Binning, both of which are known to have good performance. Both these techniques are reference-points based schemes and are efficient for overlays with a small number of peers (e.g. ≤ 32). Our results show that Tiers significantly outperforms both these schemes. Tiers is particularly efficient for large overlay networks, has an order of magnitude lower control overheads for overlays with 512 peers and still achieves greater accuracy in locating the nearest peers.

Key words: Group communication, Hierarchy, Overlays, Peer location, Triangle inequality.

1 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

* Corresponding author. Tel: +1 (608)-262 7971.
Email addresses: `suman@cs.wisc.edu` (Suman Banerjee),
`{kcr,bobby}@cs.umd.edu` (Chris Kommareddy, Bobby Bhattacharjee).

part of the application. Usually, the goal is to locate another application peer that is "near" the new host. We refer to this problem as the peer-location problem.

Efficiently locating nearby peers is an important problem for many applications. For applications like Gnutella, locating the nearest peer can reduce network load for queries and responses. In fact, nearest-peer locating techniques are applicable to a number of peer-to-peer applications. For example they are used to efficiently construct resilient overlay networks [1], implement application-layer multicast [2–5], enable efficient overlay routing [6,7], and define distributed data storage and lookup services [8,9]. Efficient solutions for nearest peer-location are also beneficial to legacy (non peer-to-peer) applications. For example, peer finding schemes can be used to locate nearby mirrors for file transfers [10], or to locate nearby sources in content distribution networks. Peer-finding schemes can naturally be used to implement application-layer anycasting services [11]. Lastly, the reachability of native multicast groups over the Internet is currently being extended by setting up dynamic unicast tunnels between multicast-enabled regions of the Internet. A solution developed for peer finding can be applied directly to create efficient tunnels. Such a solution would not only be useful for multicast, but also for efficiently deploying new overlay-based services e.g. ABone [12] and 6Bone [13].

The peer-finding problem can be solved relatively easily if we assume network-layer assistance such as native IP multicast or Global Internet Anycast (GIA [14]). Similarly, the Internet-wide distance maps system (IDMaps [15]) can also provide solutions to the peer-finding problem but it requires Internet-wide deployment of special measurement entities. Other peer-finding techniques that require a very limited infrastructure support include a triangulation method, due to Hotz [16], and its weighted variant [17], a "distributed binning" technique [18] and our prior work, Beaconing [19]. These techniques use a small set of measurement reference points in the network called landmarks [18] or beacons [19]. The distances between each application peer and these beacons are measured, and are processed to obtain the nearest peer.

In this paper, we define Tiers, a technique to efficiently solve the closest peer-location problem, without requiring any network-layer assistance or infrastructure support. Tiers is specifically designed to scale to *large* application groups and can be implemented on an unicast-only network. There are specific challenges that need to be addressed for such a unicast-only solution, which are outlined in [19].

1.1 Problem Statement

Formally, the peer finding problem can be modeled as follows:

Assume a set of peers (or hosts) H arranged in some arbitrary topology T. We denote the distance between two hosts, a and b, by $dist(a, b)$. Let A be the current set of hosts that participate in some application ($A \subseteq H$). Let $n \notin A$ be a host that wishes to join the application. The peer-finding problem finds a node p, such that $p \in A$ and $\forall q \in A, dist(p, n) \leq dist(q, n)$, i.e. p is already part of the application and is the closest such host to the new host n.

Clearly, the peer finding problem can be solved for a number of distance measures, e.g. hop count in the underlying network, application-perceived latency, etc. Tiers can be implemented with any such application-specific metric.

1.2 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.

Additionally we show in this paper that the performance of Tiers is also robust in presence of changes to group membership. One of the challenges for the closest peer-location problem is to maintain good accuracy even as peers rapidly join and leave the application group. This is an important component of any solution because application peers are typically processes on end-hosts, and are susceptible to unexpected failures.

1.3 Roadmap

The rest of the paper is structured as follows. In the next section, we describe the Tiers techniques for peer-location and present analytic bounds for this scheme. In Section 3 we describe some of the prior approaches for the closest peer location problem and how they compare to our proposed scheme. In Section 4 we evaluate and compare the performance of these different schemes through detailed simulations. Finally we conclude in Section 5.

2 Tiers : Scalable Peer Location for Large Groups

The Tiers peer-location technique arranges the set of application peers into a hierarchy. This hierarchy construction mechanism is similar to what we had defined for the NICE application-layer multicast protocol [2]. Use of this hierarchy enables scalability, since most peers are at the bottom of the hierarchy 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. 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 this paper, we use the number of hops as the distance metric between hosts.

In the rest of this section, we describe how the Tiers hierarchy is defined, the closest peer-location operation using the hierarchy, and the analytic bounds

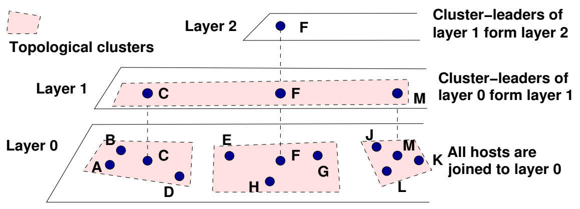
Topological clusters
Layer 2
F
Cluster−leaders of layer 1 form layer 2
Layer 1
C
F
M
Cluster−leaders of layer 0 form layer 1
Layer 0
B
A
C
D
E
F
G
H
J
M
K
L
All hosts are joined to layer 0

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

using this technique.

2.1 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 $3k-1$, 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 radius. (The radius of the cluster is defined as the maximum distance between the cluster and its members.) 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] [1] . 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.

Each peer in this hierarchy exchanges *HeartBeat* messages with each peer in all of its clusters. For example, in Figure 1, peer A sends a *HeartBeat* message to peers B, C, and D (they are all part of the same cluster in layer L_0). Similarly, C sends a *HeartBeat* message to peers A, B, D, F, and M (A, B, and D share a common cluster with C in layer L_0, while F and M share a common cluster with C in layer L_1). The lack of *HeartBeat* messages are used to detect peer

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

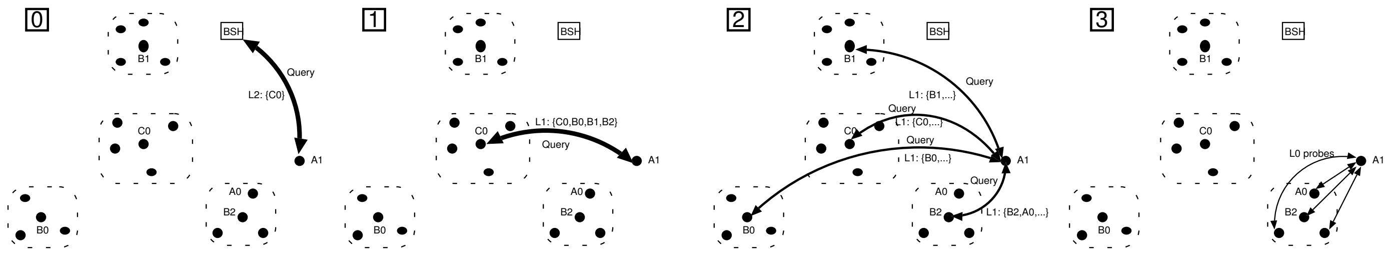

Fig. 2. Query-host A_1 finds its closest peer (A_0).

failures and trigger subsequent re-arrangement of the hierarchy.

The entire distributed protocol for this hierarchy construction (or re-structuring in case of peer failures) is same as the one employed by the NICE application-layer multicast protocol [2].

2.2 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) [2]. 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. (Alternatively it is possible that the BSH is only aware of the leader of the highest layer cluster, and is therefore not a 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, 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

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

```
Procedure : $FindClosest(h)$
$Cl_j \leftarrow Query(BSH, -)$
**while** $(j \geq 0)$
    Find $y$ such that $dist(h, y) \leq dist(h, x), x, y \in Cl_j$
    **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. $Cl_j(y)$ indicates the cluster in layer L_j to which the peer y belongs. This is defined if and only if y belongs to a cluster in layer L_j. $Query(y, j-1)$ 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.

peers (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 is presented in pseudo-code 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's cluster leader. In the worst case, the query is re-initiated from the BSH.

2.3 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 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 $3k-1$. 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 this distributed hierarchy construction protocol (which is the same as the one used to construct a similar hierarchy in NICE application-layer multicast [2]) is to scalably maintain the host hierarchy as new peers join and existing peers depart.

2.4 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 $3k-1$ peers. 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 (due to the *HeartBeats*). 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 $O(N/k^i)$. The amortized control overhead at an average peer is

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

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.

3 Existing Approaches for Peer Location

A few different approaches have been described in the prior literature that implements a peer-location service. In this discussion, let A denote the set of the peers in the application group. Let n denote the query-host.

3.1 Centralized Random Selection

In this case, all peers in A register with a well-known node w. When n decides to join the application, w chooses some node $i \in A$ uniformly at random as the "nearest" peer for n. This scheme has extremely low run-time overhead, but also incurs a high error rate. Note that Gnutella uses this heuristic to define its neighbors on the overlay. The Narada application-layer multicast protocol [4] uses this simple heuristic to define an initial set of neighbors on their mesh overlay, and subsequently refines the topology over time.

3.2 Triangulation and Weighted Triangulation

The triangulation method, due to Hotz [16], also computes distances using a set of measurement points called beacons.

Given a host i in A and k beacons $B_0, \ldots, B_{k-1}$, the triangulation method described by Hotz [16] defines a k-tuple $D_i = \langle dist(i, B_0), dist(i, B_1), \ldots, dist(i, B_{k-1})\rangle$.

A similar k-tuple $D_n = \langle dist(n, B_0), dist(n, B_1), \ldots, dist(n, B_{k-1})\rangle$ is also defined for the new node n. In the scheme, the parameter $Avg(i, n)$ is computed as

$$Avg(i, n) = \frac{Max(i, n) + Min(i, n)}{2}$$

where the Max and Min are defined as:

$$Max(i, n) = \min(|D_i + D_n|) \;\; \text{and} \;\; Min(i, n) = \max(|D_i - D_n|)$$

where the max and min are the usual arithmetic maximum and minimum computed component-wise over the k-tuples. The triangulation scheme chooses the peer i with the lowest value of $Avg(i, n)$.

Weighted triangulation [17] uses the same computation but assigns higher distances to backbone links such that after triangulation, the servers which are close to the client have a better chance of being selected. In Internet-based experiments reported in [17], the weighted scheme performs slightly better than Hotz triangulation. However, this scheme requires some way of recognizing when a distance is being measured across backbone links.

This scheme incurs high control overheads at the beacons since each beacon needs to periodically measure the distance to each of the application peers.

3.3 Expanding Ring Multicast and Broadcast

These are two classical techniques that are extremely efficient but require network-level multicast (or broadcast). In expanding ring multicast, all peers in A join a well-known multicast group. Host n too joins this group and begins sending messages to group with a low Time-to-Live (TTL). Any peer that receives this message is within a small TTL-distance of n and therefore is a close peer. If no such peer is found within a timeout period for a specific TTL, n resends the message with larger TTLs. Host n, therefore, finds its nearest peer simply by listening on the multicast group for the first response to its peer solicitation messages. This protocol is extremely robust since it does not require any centralized state at any peer and is also efficient since it does not involve any application-layer distance computations. However, as can be observed, this approach is only applicable for hop-count based distances. Additionally, the query latency can be large depending on the location of the closest peer. The expanding ring broadcast scheme is similar except all peers in the network receive the messages and only nodes in A respond.

3.4 Techniques requiring global service deployment

The peer-finding problem can effectively be solved using Global Internet Anycast [14]. All nodes in A could join an anycast group and n could find a near peer by sending a message to the anycast group. While this scheme could potentially be even more efficient than expanding ring multicast, it requires global anycast support from all participating domains. While there is an effort for standardizing the GIA protocols,such services will likely not be globally available in the near future. An Internet-measurement infrastructure such as IDMaps [15] can also be used to solve the peer finding problem. IDMaps are an Internet-wide service that provides distance information between any two nodes in the Internet. IDMaps could be used (along with a centralized directory of current peers) to find near peers or they can even be used in the beaconing protocol to find the best host in the reduced set. Unlike GIA, IDMaps do not require global changes to the network infrastructure; however, IDMaps do require Internet-wide deployment and is also unlikely to be available in the near future.

3.5 Beaconing

In this scheme, proposed in our prior work [19], we designate a set of κ hosts $(B_0, \ldots, B_{\kappa-1})$, to be reference-points in the network. These hosts are called *beacons* and are known to all peers in the application group. Each peer in the application group periodically measures the distance from itself to each of these beacons and reports this to the beacons. The beacons serve as repositories of this distance information and handle closest peer-location queries.

Consider a querying host, n, that wants to find its closest peer in the application group, A. Host n measures and sends its own distance d to a single beacon (say B_0). Beacon B_0 sends to n a list containing the identities of all hosts in A within distance $d \pm \delta$. The parameter δ is chosen such that the nearest node is within δ hops of n with high probability. Peer n repeats this procedure by measuring and sending its own distance to each other beacon B_i. The beacons respond, each with a list of peers whose distances are in the range $dist(n, B_i) \pm \delta$. Thus, h accumulates the lists of prospective *near* peers and computes an intersection of these lists. As n communicates with more beacons, the size of the intersection set reduces. The procedure is terminated when the size of the intersection set (S) is reduced to some *reasonable* number or when n has communicated with all beacons. The peers in A that are in fact close to n will typically appear in all of these sets and hence also in the intersection of the sets. Beaconing then chooses the closest peer from the set S using one of many heuristics as described in [19].

3.6 Distributed Binning

Another reference-point based closest peer-location technique has been proposed in recent literature, called Distributed Binning [18]. The goal of the Distributed Binning scheme is to have the set of peers independently partition themselves into disjoint "bins" such that peers within a single bin are relatively closer to one another than to peers not in their bin. Therefore such a scheme can be leveraged to define a good heuristic for the closest peer-finding problem (which is also discussed in [18] as a potential application).

Scheme	Infrastructure Required	Permissible Metric	Query Accuracy	Query Latency	Control Overheads
Expanding Ring Search	Network-layer multicast	Hop-count based	High	Moderate	Low
GIA [14]	Globally deployed anycast	Hop-count based	High	Low	Low
Triangulation-based [16]	Unicast-only, and deployment of few beacons	Any	Moderate	Low	High
IDMaps [15]	Unicast-only, and global deployment of tracers	Any	High	Low	Moderate
Distributed Binning [18]	Unicast-only, and existence of few landmarks	Any	Moderate	Low	High
Beaconing [19]	Unicast-only, and deployment of few beacons	Any	High	Low	High
Tiers	Unicast-only	Any	High	Low	Low

Table 1
Comparison of different peer-location schemes.

When a new peer joins the application, it computes its own bin. Subsequently the new peer locates other peers that are part of the same bin. These peers are relatively close to the new peer and are therefore appropriate candidates for being its closest peer. Each peer computes its own bin by measuring its relative distance from the set of reference-points in the network. In Distributed Binning these reference-points are called *landmark* nodes. The peer measures the round trip latency to each of these landmarks and orders the landmarks in the order of increasing round-trip times. Thus every peer has an associated ordering of landmarks. This order represents the bin that peer belongs to. The rationale behind this scheme is that topologically close peers are likely to have the same ordering for the different landmarks and hence belong to the same bin. Note that the new peer needs to be aware of the IP address of these landmark nodes and needs some out of band mechanisms to find them.

The Distributed Binning scheme incurs overheads similar to Beaconing. Each peer in the application group periodically exchanges $O(k)$ messages with the k landmarks to compute their bins. Hence each landmark receives $O(N)$ measurement (control) traffic from all the N peers in the application group.

The control overheads and query latency of Distributed Binning and Beaconing are comparable. While both these schemes are efficient for small application peer groups, they do not scale with increasing sizes. This is because the control overheads at the landmarks or beacons is $O(N)$. This can be contrasted with the Tiers approach, where the worst case control overhead at any entity is bounded by $O(\log N)$.

3.7 A Qualitative Comparison with Tiers

A key distinction of Tiers from all these prior approaches is that Tiers requires no additional deployment (of reference-points in Beaconing or Distributed Binning, or measurement entities, e.g. IDMaps) or network-level support e.g. GIA, for its operation. This is because Tiers leverages interaction between peers to construct a scalable solution to the peer-location problem. This leads to significant benefits in terms of accuracy and control overheads, with some marginal increase in query latency (typically 10-15% higher). We quantify these tradeoffs in our simulation based experiments in the next section.

In Table 1 we present a qualitative comparison of all the schemes (including Tiers).

4 Performance Analysis

We have analyzed the performance of Tiers using detailed simulations on very large network topologies. In our prior work [19], we have presented detailed evaluation of the Beaconing scheme in comparison with some other existing techniques, e.g. Hotz Triangulation [16]. Our results have showed that Beaconing achieves significant performance improvements over these other schemes. Therefore, in this paper, we restrict the performance comparisons to three schemes — Tiers, Beaconing and Distributed Binning.

Broadly, our findings can be summarized as follows: Both Tiers and Beaconing achieves better accuracy in comparison with the Distributed Binning scheme. The query latency of Tiers is marginally higher than the other two schemes (about 10-15%). However, due to its hierarchical structure, the worst case overheads at hosts in Tiers are significantly lower than both Beaconing and Distributed Binning.

4.1 Experimental Methodology

We used the Georgia Tech Internet Topology Modeler (GT-ITM [20]) to create our simulation topologies. In our simulations, we used 10,000 node Transit-Stub (TS) graphs. We experimented on fifty randomly generated TS graphs. For different experiments, we distributed upto 512 application peers uniformly at random within the stub domains. We also placed 500 query-hosts, again distributed uniformly at random, on the topology, that sought to find the closest peer using the three schemes. For both Beaconing and Distributed

Binning, we used a set of twelve beacons and landmarks respectively, as was shown to be appropriate in [18]. Also, for Beaconing, we used the Beaconing with Vectoring and Probing with $\delta = 3$ as was shown to provide good accuracy with low overheads in [19]. We ran between 10 and 20 instances for each experiment, to get a tight bound on the variations in the results. In all these experiments we use number of hops as our distance metric.

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.

Note that unlike all prior schemes, the Tiers scheme leverages distributed state management involving the application peers. As a consequence, it adds dependence between the peers of an application. Therefore in this section we also examine the impact of the effect of dynamic changes to group membership on the performance of Tiers and how it compares to that of Beaconing and Distributed Binning.

Since we assume dynamically changing application groups, we assume that each scheme incorporates a periodic soft state refresh mechanism. As defined in Section 2, the Tiers scheme already uses a periodic *HeartBeat*-based mechanism to discover peer failures. For both Beaconing and Distributed Binning, we implemented similar *HeartBeat* messages, which are sent by the application peers to the beacons or landmarks to periodically update their measured distances. For all the schemes we choose the same periodic rate (of once every 5 seconds) for these refresh messages. 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 k set to 3, 6, and 12 respectively. As mentioned, the cluster size upper bound was $3k - 1$. Such a choice of the upper bound helped in avoiding 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].

Application Peer Group Size

We varied the application group size between 8 and 512 to study its effects on the different metrics. In Figure 4 we plot the accuracy of the different schemes 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

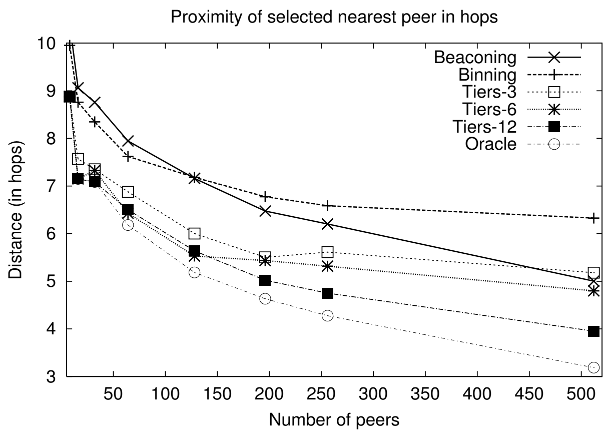

Fig. 4. Accuracy of the queries. ‘Oracle’ indicates the actual closest peer. (Varying peer group sizes).

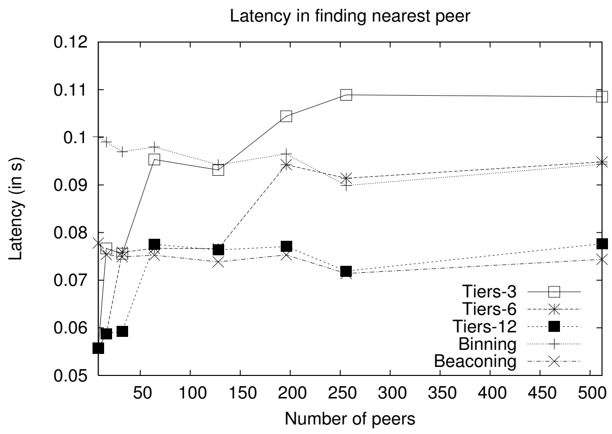

Fig. 5. Query latency (Varying peer group sizes).

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. Though the accuracy of Distributed Binning also increases with increase in the size of the peer group, it performs relatively worse in comparison to the other schemes.

In Figure 5, we plot the latency of the queries for the same set of topologies. The query latency of Beaconing and Distributed Binning does not depend on the size of the application group. In contrast, for the Tiers schemes, the query latency increases very slowly (logarithmically) with the increase in peer group size. In Figure 5, we can observe that 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

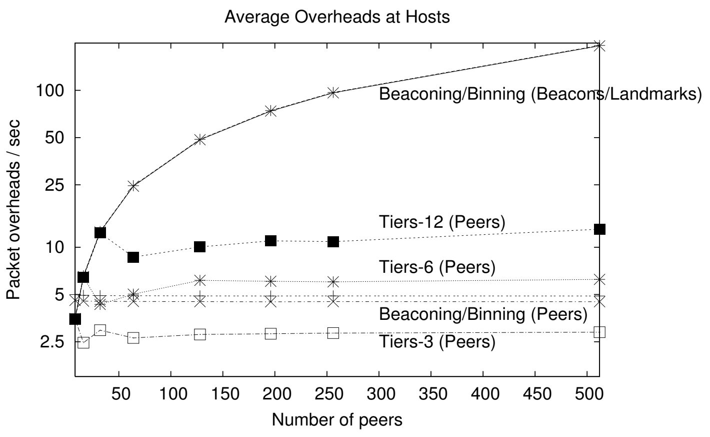

Fig. 6. Average control overheads at the end-hosts.

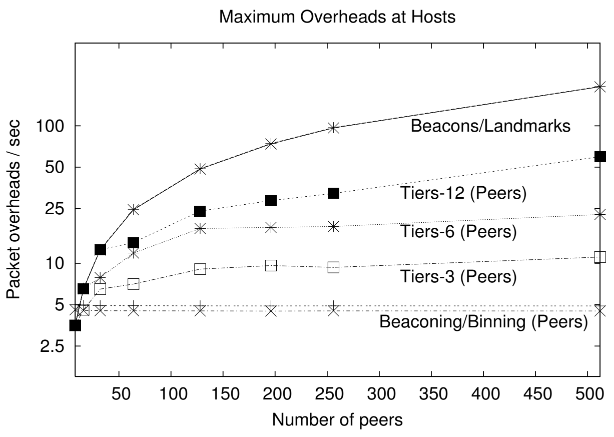

Fig. 7. Worst case control overheads at the end-hosts.

actually decreases (sometimes marginally) 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.

In Figures 6 and 7 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 peers for Beaconing and Distributed Binning are very close to each other. Among the different Tiers schemes, the peers in the Tiers-3 has the lowest overheads (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) and the landmarks (in Distributed Binning) 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 and is about an order of magnitude lower (22.8 packets/second) for the peer groups of size 512. The

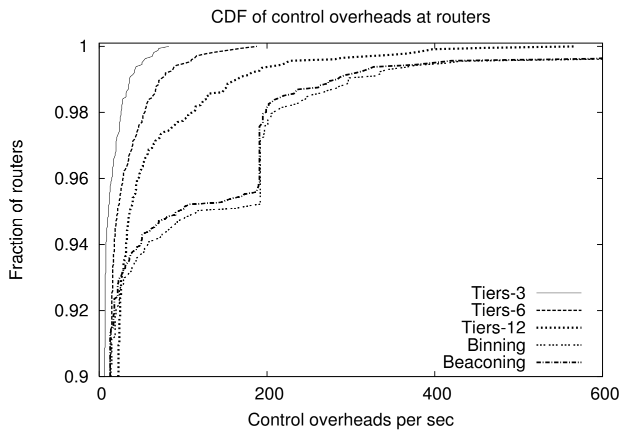

Fig. 8. Distribution of control packet overheads at the network routers.

overheads at the beacons increase linearly with the peer group size, where as the worst case overheads for Tiers increase logarithmically. Therefore, both Beaconing and Distributed Binning are efficient and fast for small groups, but does not scale with increasing group sizes.

Finally in Figure 8 we examine the control overheads incurred at the different network routers in the topology for the different schemes for a group of 512 peers. The figure plots the cumulative distribution of the control packets at the different network routers. For all the schemes, the overheads incurred at most routers are low (about 94% of the routers carry less than 20 packets per second). However, the network routers close to the beacons in Beaconing and Landmarks/central entity in Distributed Binning carry high volume of data.

The worst case control overheads at the routers for Beaconing and Distributed Binning are close to 1000 packets per second. In contrast, for the Tiers-3, -6, and -12 schemes, the worst case is less than 100, 200, and 600 packets per second respectively.

Effect of the k parameter

In the Tiers scheme, the different metrics of interest —- query accuracy, query latency and control overheads, are affected by the choice of the cluster size lower bound parameter, k. In Table 2 we examine this dependence more closely. Note that if k is chosen to be the size of the application peer group, N, then the Tiers scheme has a single layer with a single cluster. In such a case the scheme degenerates to explicit probing of all the application peers by the query-host. Clearly this scheme will be perfectly accurate. The control overheads in this scenario will be high ($O(N)$ at each peer). However, the query latency will be minimum, since all the distance probes will occur in parallel. Therefore in

Scheme	Accuracy	Latency	Overheads at host	
			Average case	Worst case
Oracle	3.33	-	-	-
Distributed Binning	6.48	84.3	4.88	192.87
Beaconing	5.14	69.6	4.49	191.59
Tiers-k				
k = 3	5.05	103.0	2.94	11.42
k = 4	4.79	105.4	4.10	14.59
k = 5	4.61	84.8	5.04	16.35
k = 6	4.55	85.7	6.11	22.38
k = 7	4.70	88.2	7.31	25.09
k = 8	4.31	89.2	8.36	27.77
k = 9	4.42	87.0	9.30	39.53
k = 10	4.13	84.8	9.36	35.33
k = 11	4.05	68.7	12.24	55.09
k = 12	3.93	68.6	12.78	55.12

Table 2
Effect of the cluster size lower bound parameter, k, for the Tiers scheme on an application group with 512 members.

general, as the value of k increases, the accuracy of the scheme in finding the closest peer increases, the control overheads increase, and the query latency decreases. We see this trend in Table 2. Also note that the query latency does not decrease smoothly with increase in the cluster lower bound parameter, k. The latency only decreases when the number of layers in the hierarchy decreases. This happens as k increases from 4 to 5, and from 10 to 11.

Therefore, this parameter can be appropriately chosen to trade-off between the protocol performance and control overheads.

Changing Group Membership

Changes in membership of application peer groups will change the closest peer to a query host. In this part of the study, we examined how adaptive the different schemes are to such dynamics of group membership.

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

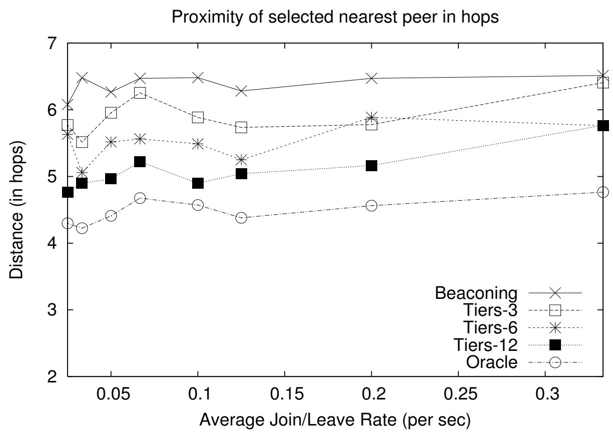

Fig. 9. Accuracy of queries. 'Oracle' indicates the actual closest peer. (Varying join/leave rates).

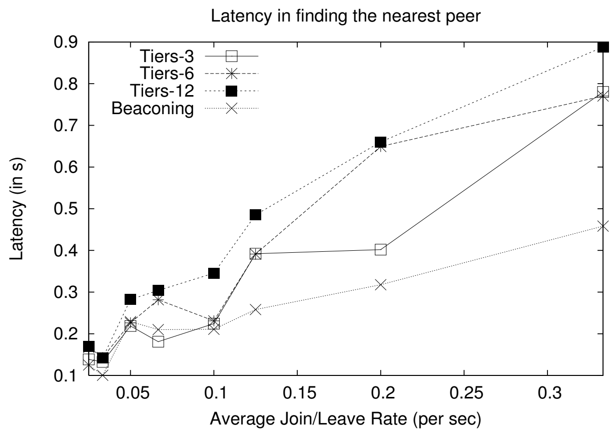

Fig. 10. Query latency (Varying join/leave rates).

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 9, 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 10). 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 performance of Beaconing and Distributed Binning in these experiments had similar trends. So for the sake of clarity, we only included the data for the Beaconing scheme in the plots.) 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.

5 Conclusions

In this paper, we have presented Tiers, a new scheme for scalable peer location on the Internet. While simple alternative techniques like Beaconing and Distributed Binning, performs well for small groups, a hierarchical approach like Tiers is essential to scale with increase in group sizes. Through detailed simulations we show that Tiers achieves similar performance with significantly lower overheads than any other scheme for groups larger than 32. The protocol is based on a hierarchical clustering of the peers.

While in this paper, we describe the protocol to find the closest peer with respect to latency-based metrics, it has a wider applicability to other metrics as well. 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. Andersen, H. Balakrishnan, M. Frans Kaashoek, R. Morris, Resilient overlay networks, in: Proceedings of 18th ACM Symposium on Operating Systems Principles, 2001.

[2] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable Application Layer Multicast, in: Proceedings of ACM Sigcomm, 2002.

[3] P. Francis, Yoid: Extending the Multicast Internet Architecture, white paper http://www.aciri.org/yoid/ (1999).

[4] Y.-H. Chu, S. G. Rao, H. Zhang, A Case for End System Multicast, in: Proceedings of ACM SIGMETRICS, 2000.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, SCRIBE: A large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in communications (JSAC) 20 (8).

[6] B. Y. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Tech. rep., UCB/CSD-01-1141, University of California, Berkeley, CA, USA (Apr. 2001).

[7] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.

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

[9] S. Ratnaswamy, P. Francis, M. Handley, K. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of SIGCOMM, 2001.

[10] E. Zegura, M. Ammar, Z. Fei, S. Bhattacharjee, Application-layer anycasting: a server selection architecture and use in a replicated web service, IEEE/ACM Transactions on Networking 8 (4).

[11] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, Z. Fei, Application-Layer Anycasting, in: Proceedings of INFOCOM 97, 1997.

[12] Abone: The Active Backbone, see `http://www.isi.edu/abone`.

[13] 6bone: The IPv6 Backbone, see `http://www.6bone.net`.

[14] D. Katabi, J. Wroclawski, A framework for scalable global IP-anycast (GIA), in: Proceedings of ACM SIGCOMM, 2000.

[15] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, Y. Jin, An architecture for a global Internet host distance estimation service, in: Proceedings of INFOCOM, 1999.

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

[17] J. Guyton, M. Schwartz, Locating nearby copies of replicated Internet servers, in: Proceedings of SIGCOMM, 1995.

[18] S. Ratnaswamy, M. Handley, K. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: Proceedings of INFOCOM, 2002.

[19] C. Kommareddy, N. Shankar, B. Bhattacharjee, Finding close friends on the Internet, in: Proceedings of ICNP, 2001.

[20] K. Calvert, E. Zegura, S. Bhattacharjee, How to Model an Internetwork, in: Proceedings of IEEE Infocom, 1996.

Suman Banerjee received his Ph.D. and M.S. in Computer Science from University of Maryland in 2003 and 1999 respectively, and his B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kanpur, India in 1996. Since 2003, he has been an assistant professor in the Department of Computer Sciences at the University of Wisconsin-Madison. His research interests are in protocols and services for networking and distributed systems.

Christopher Kommareddy received his B.E.(Hons) degree in Electrical Engineering and M.Sc. in Chemistry from BITS, India in 1999. He is currently pursuing his Ph.D. in Electrical and Computer Engineering at the University of Maryland. From 1998 to 1999, he interned at Tata Elxsi, Bangalore, India and in 2000, at Hughes Network Systems, Maryland, USA. He is currently a research assistant with the UMIACS. His research interests include network architecture, routing, network protocols and overlay networks.

Bobby Bhattacharjee received his Ph. D. in Computer Science from Georgia Tech. in 1999. Since 1999, he has been an assistant professor in the Computer Science department at the University of Maryland, College Park. His research interests are in network protocols, network security, and distributed systems. He is a member of the ACM.