Scalable Self-Organizing Overlays

Sushant Jain, Ratul Mahajan, David Wetherall, and Gaetano Borriello
{*sushjain,ratul,djw,gaetano*}*@cs.washington.edu*

Technical Report UW-CSE 02-06-04
Computer Science and Engineering
University of Washington

Abstract—

Recent developments including peer-to-peer networks and application-level multicast have generated interest in overlays, and by extension in algorithms for automatically maintaining overlay structures. In this paper, we present a self-organizing protocol for forming and managing overlays that scales to at least tens of thousands of nodes. Previous algorithms target small scale overlays and do not scale well in terms of protocol overhead with the number of nodes. Our key contribution is to show how to apply the concept of hierarchy to overlay management to achieve scalability, without degrading the quality of the resulting overlay. We provide simulation results that show our hierarchical overlays incur a low delay penalty, make effective use of network bandwidth, and handle failures gracefully. We also demonstrate that there are significant advantages in using a self-organizing overlay as compared to the ad hoc techniques currently being used in peer-to-peer networking.

I. Introduction

Overlays are used as a mechanism to deploy new distributed applications and protocols on top of the Internet. Examples include the MBone [1], the ABone [2], the 6Bone [3] and Gnutella [4]. An overlay is formed by a subset of nodes drawn from an underlying network. Participating nodes communicate through *tunnels*, which are virtual links between two nodes that may not be directly connected in the underlying network. These tunnels define the topology of the overlay. A self-organizing overlay protocol maintains an efficient and connected topology when the underlying network fails, performance changes, or nodes join and leave the network.

Most currently deployed overlays are statically configured. This is a straightforward technique, but possesses several disadvantages compared to automatic configuration using a self-organizing overlay protocol. Static configuration has a high management overhead that quickly becomes unwieldy when the overlay grows beyond a small number of nodes. Gnutella, in contrast, has grown to over a thousand nodes [5], because it is not statically configured. Static configuration can fail to provide connectivity in a dynamic environment where nodes are free to join and leave the overlay at will, as is again the case in peer-to-peer networks like Gnutella. Finally, static configuration misses the opportunity to improve overlay performance by measuring and adapting to changing network conditions. For these reasons, self-organizing overlay protocols are likely to displace static configuration methods in the near future.

Prior work like Narada [6], Gossamer [7], Overcast [8] has presented self-organizing protocols that are suitable for overlays with up to a few hundred nodes. Yet the success of applications such as Gnutella demonstrates a clear need for much larger overlays. The primary contribution of our work is a new self-organizing protocol that scales to at least tens of thousands of nodes without sacrificing the quality of the overlay. We achieve this scalability by exploiting the concept of hierarchy. Our overlay is organized in a two-level hierarchy and we present mechanisms to create and manage this hierarchy dynamically. As a secondary contribution we also present enhancements to the Narada algorithm [6], which is used by our protocol.

We demonstrate that using hierarchy has minimal effect on performance. At the same time hierarchy achieves scalability by drastically reducing the bandwidth requirements for overlay maintenance. We also show that hierarchy helps in mitigating the effect of a dynamic environment; most changes are not seen beyond a small set of nodes. It is particularly well-suited to heterogenous environment like the Internet when the upper level nodes in hierarchy are chosen carefully.

We believe that using a two level hierarchy we can scale to tens of thousands of nodes, an order of magnitude more than existing schemes. Given the size (of around a thousand) of many overlay networks like the MBone, the 6Bone, and Gnutella, scaling to tens of thousands of nodes is a reasonable first step towards more scalable overlay protocols. The ideas presented here can be extended to multiple levels of hierarchy to achieve further scalability.

Our overlay management protocol is broadly targeted at application level multicast, test-beds for new protocols, active network infrastructure, resource discovery, content distribution, and so forth, rather than a particular applica-

tion. That is, our intent is to provide an efficient and robust overlay management protocol that can be used by variety of applications. We believe that for many applications the advantages of using the generic overlay protocol will outweigh the cost of designing one for each application in the same manner that shortest-path routing has proved to be a suitable base for many different applications. Furthermore, the ideas presented in this paper - hierarchy and its evaluation - can be extended to design self-organizing overlay protocols tuned towards a particular application.

The rest of the paper is organized as follows. In the next section, we describe classes of applications that use overlays and survey the existing work in those classes. Section II discusses how to evaluate various overlay topologies. Section IV describes our new protocol. In Section V, we evaluate our protocol using simulation. Section VI shows the performance of our overlays in the context of broadcast-based resource discovery application like Gnutella. We conclude in Section VII.

II. Overlay Applications and Related Work

Overlays have been used/proposed for a variety of applications. This section provides a brief overview of both the applications and the existing overlay construction proposals, and talks about how they compare to the protocol proposed in this paper.

Application-level multicast is made possible through the use of overlays. The proposed solutions can be divided into two classes based upon their approach. Using what is called the *tree-first* approach, Yoid [9] and Overcast [8] form the multicast tree directly. In a *mesh-first* approach nodes are connected in a mesh (not a fully-connected one), and the multicast tree is formed by running a multicast routing protocol like DVMRP [10] on top of the mesh. Examples of this approach are Narada [6], and Gossamer [7]. A single-level mesh does not scale beyond a few hundred nodes [6]. We use a two-level hierarchy to buy scalability, with a mesh-first approach at both levels.

Another class of applications use overlays for wide area broadcast-based resource discovery. Gnutella [4] is one such large, completely decentralized overlay. It is primarily used to broadcast a search query from one member to all the other members of the overlay. Tunnels are selected randomly in Gnutella so the overlay formed is inefficient. In Section VI, we will compare overlays formed by our protocol with such overlays. The Intentional Naming System (INS) [11] proposes a resource discovery solution using an overlay in which a spanning tree is configured based upon latency. A tree topology, for general purpose overlays, can be fragile and inefficient. It also imposes high overhead at nodes higher up in the tree, and hence is not appropriate for a large scale dynamic overlays.

Overlays are also used in content distribution networks. Research in such overlays focuses on the problem of distribution and location of content inside the overlay. Several distributed indexing approaches (CAN [12], Chord [13], Pastry [14], Tapestry [15]) have been proposed to address the problem of locating content in a large distributed system. These systems locate content by routing queries across a geometric structure. This is realized as an overlay and the challenge is to efficiently map this overlay on top of the Internet. These schemes are application (locating a resource given its hash key) specific, and do not attempt to make efficient overlays usable for a large class of applications, a goal of our work. For instance, in a resource discovery application in which resources are not well-specified (like Gnutella), one has to resort to broadcasting the query, an operation supported by our overlays.

Incremental deployment of new protocols was one of the first uses of overlays. Examples include the MBone [1], the ABone [2], and the 6Bone [3]. All of these overlays are configured statically.

III. Overlay Performance Metrics

We use multicast as a representative driving application to quantify the efficiency of an overlay. Several recent proposals have discussed overlay management in the context of application-level multicast [6], [8], [7]; the proposals typically construct multicast distribution trees rooted at the source of a multicast transmission. Regardless of the specific details of each particular proposal, two metrics that can be used to evaluate performance are *relative delay penalty (RDP)* and *stress*.

RDP is a measure of the additional packet delay introduced by overlay on the delivery of a single packet between a source and destination. More specifically, RDP is the ratio of the latency experienced when sending data using the overlay to the latency experienced when sending data directly using the underlying network.

Stress is a measure of the excess bandwidth consumption induced by the overlay during a multicast transmission. The stress of a physical link is defined as the number of overlay tunnels that send traffic over that link. Note that stress is both a function of the topology and the multicast tree used: flooding-style broadcasts cause more stress on a physical link than multicasts. For efficient multicast trees, a multicast packet flows over each virtual overlay tunnel at most once.

Ideally, an overlay should have both low RDP and low stress. Unfortunately, these requirements can conflicting. To see this, consider Figure 1(a), which shows an example physical network with four hosts interested in forming

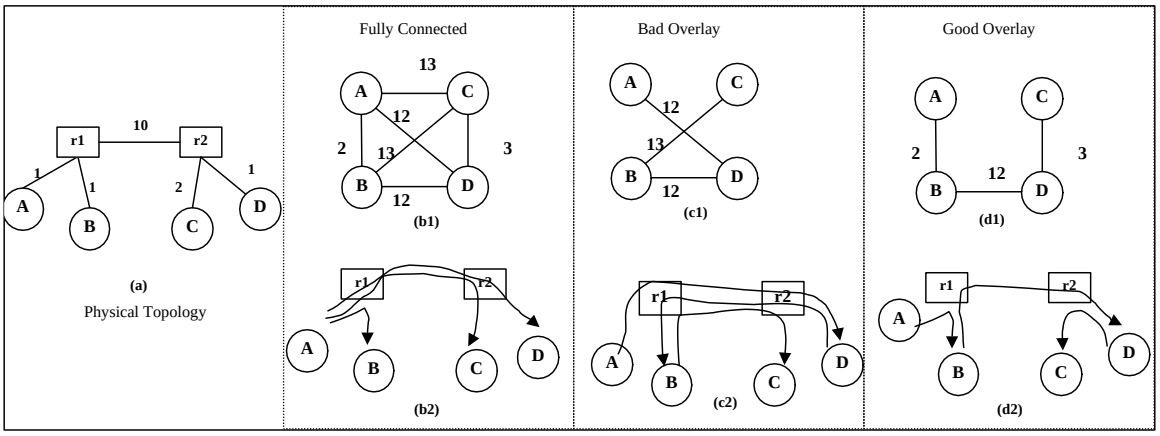

Fig. 1. Example illustrating different overlay topologies. (a) shows a physical network with 4 hosts interested in participating in overlay. Three different overlays topologies are shown. Below them is shown how A would broadcast data to B,C,D using the overlay.

an overlay. Figure 1(b1) shows a fully connected overlay topology; in this case the RDP between all pairs is 1, since they send to each other directly. However, the stress on links close to end hosts is high: Figure 1(b2) shows the paths taken by packets if A wants to communicate with all B, C, and D simultaneously. A must send three packets over its physical access link, one per overlay tunnel, leading to a stress of 3 on that physical link.

As another example, consider an overlay topology that selects overlay links randomly, resulting perhaps in the overlay shown in Figure 1(c1). In this case, all overlay tunnels go over the physical link R1-R2, leading to high stress. Additionally, the RDP between most pairs of nodes is very high (37/13 for (A,C), 24/2 for (A,B) and 25/2 for (C,D)).

A common approach to counter high link stresses, while keeping RDP's low is to place a bound on the outdegree of the nodes and select the best possible tunnels within that degree bound. The optimal graph in this case is a *degree-bounded K-spanner* [16], in which RDP between any two nodes is less than K. The problem is NP complete, however heuristics can be used to obtain reasonably good overlays [6], [7]. Our protocol also uses this approach and extend the ideas presented in Narada [6]. A good quality overlay with low RDP's and low stresses is shown in Figure 1 (d1).

IV. The Protocol

In our approach, the overlay topology is a two-level hierarchy. The topology is partitioned into clusters. A cluster is a set of nodes. Every cluster has a unique representative node called the *head*. The rest of the nodes are called *child* nodes. Any data entering or leaving the cluster goes through the head node. Figure 2 illustrates the two-level hierarchy.

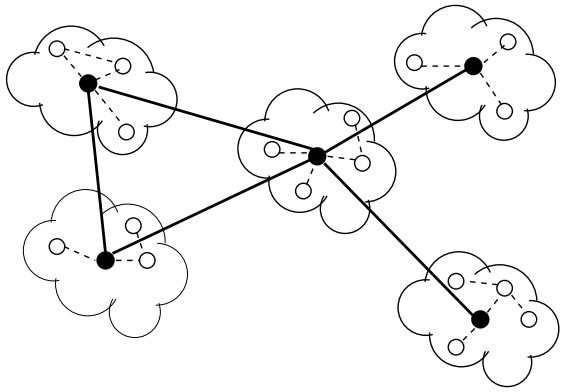

Fig. 2. The Two-level hierarchy: clouds represent clusters with the solid nodes as head nodes. The solid lines between the heads is the top level topology.

The use of a two-level hierarchy divides the overlay construction task into two independent sub-problems: clustering and mesh management. Clustering deals with forming the clusters. Mesh management determines how the nodes at the same level are connected to each other. It comes into play at both levels of hierarchy; it determines how the nodes in the same cluster are connected to each other, and how the head nodes connect to each other (also called the *top-level* topology). In a flat overlay, mesh management is the only protocol required, and thus, is a complete self-organizing overlay protocol in itself like Narada [6] and Gossamer [7]. Thus hierarchy serves two purposes: it increases scalability since measurement probes are run across smaller groups (discussed in Section IV-C.3) and it decreases management overhead by localizing the effects of member failures within smaller groups. However, a hierarchy potentially loses some opportunity for efficiency, since child nodes in different clusters do not have the abil-

ity to form overlay links to each other.

A node wishing to join the overlay obtains the address of node(s) in the overlay through the DNS or some other bootstrapping mechanism. It then randomly picks a cluster to join and becomes a part of the overlay immediately. Over time, the position of the node is improved by the protocol. In the following sections, we describe in detail how mesh management and clustering work, followed by a discussion of other aspects like scalability and behavior under dynamic conditions.

In our design we have chosen to optimize latency, the overhead of transmitting data using the overlay. Bandwidth is another obvious metric of interest. The decision to use latency was guided by the following observations. 1) Measurement of bandwidth is less reliable; for instance, two overlays nodes see the same available bandwidth over a physical link when they measure, but each of them would actually get half of it if they send simultaneously. 2) It is hard to evaluate the resulting bandwidth properties because they are very application specific (unicast v.s. multicast, for instance). As mentioned in Section II, to protect against overloading physical links in absence of bandwidth based optimization we place a bound on the outdegree of an overlay node.

A. Mesh Management

The mesh management protocol selects the tunnels that connect the nodes at the same level. When choosing these tunnels, there is a tension between overlay performance and efficient utilization of underlying network. Performance is determined by the latency overhead of routing data using the overlay; a fully-connected mesh is most efficient in this regard, but leads to an inefficient use of underlying network because several tunnels would go over the same physical link.

Instead of designing a mesh management protocol from scratch, we extend Narada [6]. In Narada, the participating members periodically perform a set of operations, which improves the overlay over time and keeps it connected. We now describe these operations and point out our extensions. We evaluate the effect of our extensions to Narada in Section V-B.1.

A.1 Mesh Management Operations

- *Add:* Nodes add tunnels to other nodes when the utility of the tunnel exceeds a threshold. The utility of a tunnel, computed as shown in Figure 3, is dependent on the extent to which it improves the node's latency to other nodes in the overlay. Tunnels are not added by a node that has a high degree until it deletes some of its tunnels. This operation is unchanged from Narada.

$utility(i, j)$
$utility = 0$
for each member m at the same level
  CL = current latency between j and m along mesh
  NL = new latency between j and m if the link $i - j$ is added
  if $(NL < CL)$
    $utility += 1 - (NL/CL)$
return $utility$

Fig. 3. Utility of Link (i, j)

- *Delete:* An existing tunnel is deleted when it is no longer useful, which can happen with changing node membership and network conditions. The principal idea behind this operation is that an existing tunnel that would not be added by the add operation, if it was removed, should be deleted. To test whether an existing tunnel can be deleted, the node first determines if there is an alternate path available to the remote node. This is done by examining the existence of a path to a neighbor of the remote node that does not go through the remote node itself. In the absence of such a path, the tunnel is not deleted, as it might partition the overlay. If such a path exists, the utility of this tunnel is evaluated by assuming that the tunnel does not exist, and then evaluating the utility of the tunnel just like in the add link operation above. The tunnel is deleted if its utility is lower than a threshold.

The delete operation described above is different from that in Narada. Narada estimates the deletion utility of a tunnel as the number of nodes the tunnel is used to reach and uses a lower threshold as a way to prevent partitions. These heuristics make the link deletion harder in Narada. The symmetry between delete and add operations in our modified version of the delete operation leads to more effective deletion, making it easier to remove less useful tunnels.

- *Swap:* With just the above two operations, new links with good utility may not be added because one of the involved nodes already has its maximum number of tunnels, each of which is above the deletion threshold. The swap operation exchanges a tunnel with lower utility for one with higher utility. Swapping helps nodes to pick the best tunnels, independent of the order in which they join the overlay or the order in which they try adding tunnels to other overlay nodes. Swapping is not present in Narada.
- *Partition detection and repair:* This operation maintains the connectivity of the overlay in the face of failures. Partitions are detected using refresh messages. In a connected network members send and receive refresh messages. If the overlay is partitioned, then members will not be able to hear refresh messages from some node, indicating a parti-

tion. Once detected, the partition can be repaired by trying to add links to such nodes. Details for this operation can be found in [6].

B. Clustering

The objective of clustering is to maintain appropriate clusters, both in proximity of nodes inside a cluster, and the cluster size. We use latency as the basis to form clusters by grouping together nodes that are close to the head of the cluster. Having a close head would induce minimal communication overhead on the child nodes. Moreover, if the children are close to the head, the children themselves are expected to be close to each other, which leads to low overhead of maintaining the cluster overlay topology.

The appropriate size of clusters is dependent on the size of the overlay and the application. For example, for small overlays (say 100 nodes) no clustering might be required. However for an overlay with 10,000 nodes the protocol would not scale without clustering. We specify the desirable cluster size as *expected cluster size* $ECS(n)$, a function of the overlay size n. Having a function of n is different from having a fixed parameter because it enables the protocol to adjust the amount of clustering as the overlay size grows or shrinks. We now describe the various clustering operations.

B.1 Clustering Operations

There are three clustering operations. The first one, *migrate*, helps a node to move to the cluster closest to it. The remaining two, *split* and *diffuse*, keep the cluster sizes within the desired range.

Migrate: This operation moves a child node from one cluster to another. Migrate tries to place a node in the cluster whose head is closest to that node. When a child joins the cluster, it receives information about other cluster heads from its current cluster head. (Each cluster head periodically broadcasts the list of other heads to its children using the bottom level overlay). The child then periodically probes a small number of carefully selected heads to determine its latency to them. Once the child detects a head node which is significantly closer to it than its current head, it leaves its current cluster and joins the new cluster. To avoid multiple migrations immediately after a node joins the overlay (as the initial joining cluster is picked randomly), a new node does not migrate until it has measured its latency to a significant fraction of other head nodes.

Migrate requires child nodes to probe all head nodes, which can overload head nodes in a large overlay. Heuristics can be used to reduce these probes. For example, in practice a child does not need to probe a head node that is more than $2 \times \mu$ from the child's head, where μ is the latency between the child and its head [1]. In our experiments, this heuristic reduces the number of probes sent by up to 50%. Another possibility is to use *landmark ordering* [12] to choose appropriate cluster. In landmark ordering, every node (both child and head) measures its latency to a predefined set of landmark nodes and orders them. A child joins the head whose ordering most closely matches its own.

When a child node migrates to a new cluster, all the previous mesh management tunnels are deleted for that child. After joining the new cluster, the new head node randomly assigns it a neighbor to bootstrap the lower level mesh management.

Split: A cluster that is more than twice as large as $ECS(n)$ in size is broken into two. The decision to split a cluster is taken by its head node. After a split, the old cluster head remains as the head of one of the new clusters, but a new head must be selected for the second new cluster. The new cluster head is chosen to minimize the average latency to all other child nodes in its cluster; this decision requires knowledge of the latency between all pairs of child nodes.

Fortunately, every child node knows its latency to all other child nodes as a side-effect of the mesh management protocol. Whenever a decision to split occurs, the old cluster head broadcasts its latency to its child nodes. On receiving this information, each child node computes the number of other child nodes that are closer to than the old cluster head, and each child reports this number back to the old cluster head. After receiving enough reports, the old cluster head selects the new cluster head. The new head creates a tunnel to the old head, and from there on mesh-management enables it to get an appropriate set of tunnels to other cluster heads. The child nodes closer to the new head are informed of the split and they migrate to the new cluster.

Head selection in our implementation is based upon latency, however one can imagine incorporating other criteria, such as stability or access link bandwidth.

Diffuse: Over time, clusters may diminish in size because of node migrations and node deaths. Clusters that are more than a factor of two smaller than $ECS(n)$ are disbanded altogether, i.e., are diffused. All of the nodes in a cluster undergoing diffusion are migrated to neighboring clusters. To avoid the possibility of partitioning the top-level topology when the head node of the diffusing cluster migrates, links from its new head are made to all of its previous neighbors. Unnecessary links are eventually deleted

[1] This is assuming absence of Detour[17] affect

by mesh management.

C. Discussion

C.1 Behavior under Dynamic Environment

Hierarchy has significant advantages when it comes to operating in dynamic environments. Joining the overlay is a very simple operation, and does not effect nodes beyond the cluster the new node joins in. Similarly, death of a child node is of interest only within the cluster. Thus, the transient behavior of a child node has a very localized effect. Since most nodes are child nodes, the resulting overlay structure is highly robust towards dynamic nature of the participating nodes. On the other hand, in a flat overlay every change in membership is propagated to every other member and hence causes a lot more disruption. We present simulation results about this in Section V-C.2.

C.2 Exploiting Heterogeneity

Resilience of hierarchical overlays towards dynamic environment can be multiplied manifold by exploiting heterogeneity present among overlay nodes [5]. As discussed above, the membership of child nodes have very limited effect on the overlay. It is the head nodes that matter; chosing them carefully significantly increases the stability of the overlay as a whole.

When nodes inside a cluster are homogeneous, choice of the head node is not that important. When nodes are heterogenous, the important factors include the access bandwidth (modem or broadband) of the node, as well as the stability of the node. Nodes that tend to remain in the overlay for longer periods should be preferred. Well-connected head nodes can take the extra load imposed on them due to hierarchy (managing their cluster); stable heads minimizes the effect of dynamic membership on the overlay (Section IV-C.1); and the potentially poorly-connected or unstable participants are shielded from the bulk of overlay management tasks as they tend to remain as child nodes (see Section V-C.1 for details on bandwidth requirement of control traffic of the two categories of nodes).

C.3 Scalability

By itself the mesh management protocol does not scale beyond a few hundreds of nodes [6]. Mesh management requires every member of the overlay to probe and maintain state for all the members of the overlay. This is inherently unscalable because: 1) traffic generated for protocol maintenance grows significantly with overlay size (Section V-C.1); 2) time taken to adapt to changes increases rapidly with the overlay size (Section V-C.2). Hierarchical overlays avoid both penalties as a result of information hiding achieved due to clustering.

V. Evaluation

In this section we evaluate our protocol using simulation. The primary goal of the is to compare hierarchical overlays with flat ones along the dimensions of interest. The question we try to answer in this comparative evaluation are:

1. **Latency Overhead:** There is a latency overhead in using the overlay instead of the underlying network. How does this overhead change when moving from a flat overlay to a hierarchical overlay? (Sections V-B.2 & V-B.3)
2. **Protocol Overhead:** What is control traffic overhead of maintaing a hierarchical overlay compared to a flat overlay? To what extent does hierarchy help in this regard? (Section V-C.1)
3. **Dyanamic Environment:** Which type of overlays are more suited towards dynamic environment like unstable participation of nodes? (Section V-C.2)
4. **Load Balancing:** In a hierarchy, there is greater responsibility on head nodes. How is the overall responsibility divided among head nodes? (Section V-C.3)

Apart from the above, we also provide results on the following:

1. **Improved Mesh Management:** How much improvement do our extensions to Narada provide? (Section V-B.1)
2. **Recovery from failures:** How is the connectivity of the overlay affected due to node failures and how quickly does the overlay recover? (Section V-D)
3. **Resource Discovery:** In the context of a specific application of broadcast-based resource discovery, how much better do our overlays perform compared to ad hoc techniques like Gnutella? (Section VI)

A. Simulation Setup

We have written a custom event driven simulator to simulate our protocol. We have not simulated dynamic network conditions like queueing delay, packet losses, congestion or varying latencies. The first few nodes are made head nodes, and the rest of the nodes bootstrap from the list of these head nodes. Head nodes exchange routing messages every 30 seconds. The frequency of the mesh management operations is 10 seconds. Child node probes other heads once every 10 seconds. In mesh management outdegree of a node is bounded by 8. Simulations are run until there are no topological changes for about 15 minutes of simulated time.

We use the Georgia Tech Internetwork Topology Models[18] (GT-ITM) to generate the network topologies used

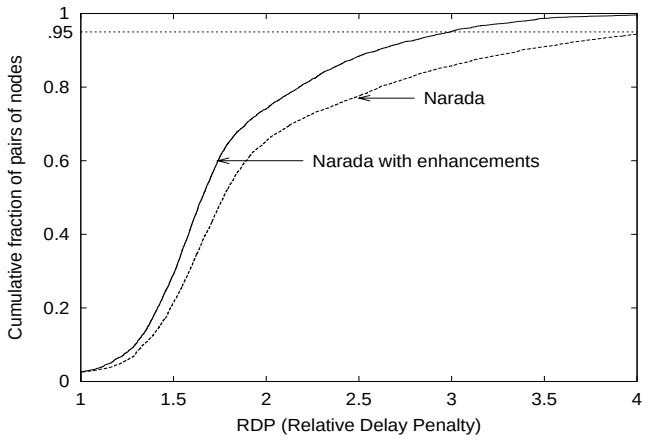

Fig. 4. Effect of extensions to Narada on RDP

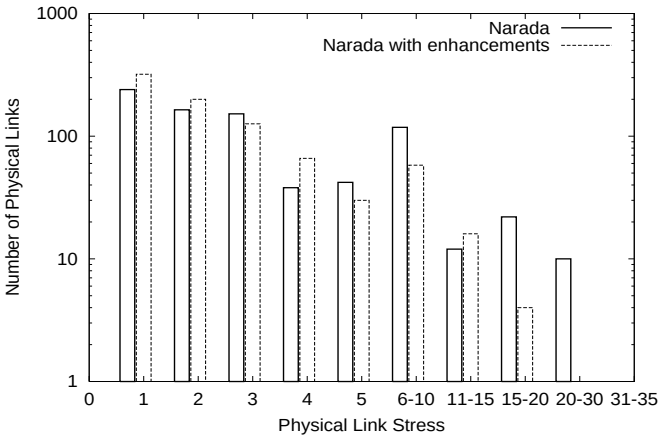

Fig. 5. Effect of extensions to Narada on stress of a physical link

in our simulations.[2] We use the "transit-stub" model to obtain graphs that more closely resemble the Internet hierarchy than a pure random graph. Unless otherwise specified, a topology of 12,800 nodes is used for the simulations (we have obtained similar results for other underlying topologies). It consists of 4 transit domains, each with an average of 8 stub networks. Each stub network has 4 stub domains each with around 100 nodes. Latencies to links in the physical topology are assigned by GT-ITM. Nodes that become part of the overlay are chosen randomly from those in the stub domain.

B. Latency

We quantify the latency overhead of using the overlay using *Relative Delay Penalty (RDP)*. Recall from Section II that RDP is the ratio of the overlay latency to the physical latency between two nodes. To summarize the simulation results for an overlay we will also use the 95% (percentile) RDP, which is the penalty seen by the 95% of the pairs of nodes. In Section V-B.2, we show that extremely high RDPs are mostly associated with pairs of nodes having small physical latency, hence the 95% RDP helps to remove these tail-effects. Another advantage of using the 95% RDP is that it is fairly insensitive to simulation parameters compared to the maximum RDP.

B.1 Improvement over Narada

In order to see the effect of extensions we made to Narada, we simulated a flat overlay (no hierarchy) with 200 nodes, without and with our extensions (symmetric deletion and swapping). An underlying topology of 3600 nodes was used. Figure 4 shows that our extensions lead to significantly lower latency overhead. With original Narada, the 95% RDP was 4.1; with extended Narada it was reduced to 2.9. The maximum RDP in the former is more than 7, while it is about 4 in the latter. Note that with clustering, the benefits of improvement at the top level are magnified because child nodes route through their respective head nodes.

Figure 5 shows that the improvements over Narada, do not come at the cost of excess load on underlying physical links. The graph compares the histograms of *link stress*, which is the number of overlay tunnels that go over a physical link in the underlying topology. Physical links with high stress are more likely to be overloaded when the overlay is being used. These results are from a single simulation run, but the qualitative comparison between the two histograms is typical. There is slight overall improvement with our extensions in the link stress, and in general we observe lower worst case stress (20 compared to 27 in this case).

B.2 Overlay Size

To observe the effect of overlay size on RDP, we increased the overlay size from 300 to 10,000. The expected cluster size is chosen so that the number of head nodes stay roughly the same across different overlay sizes, 300 in this case (the simulation with 300 nodes is a flat overlay).

Figure 6 shows the cumulative distribution of RDP of overlays of various sizes. There is very little degradation in RDP as the overlay size increases. Averaged over many simulations, the 95% RDP for 300 nodes was 2.9, and for 10,000 nodes was 3.05. This is very encouraging since it suggests that with clustering one can scale to large overlays without suffering on RDP. The primary reason behind

[2] We have recently begun to re-run our simulations using "power-law" topology generators such as Brite and Inet. While we have not completed this task, our preliminary results show similar results.

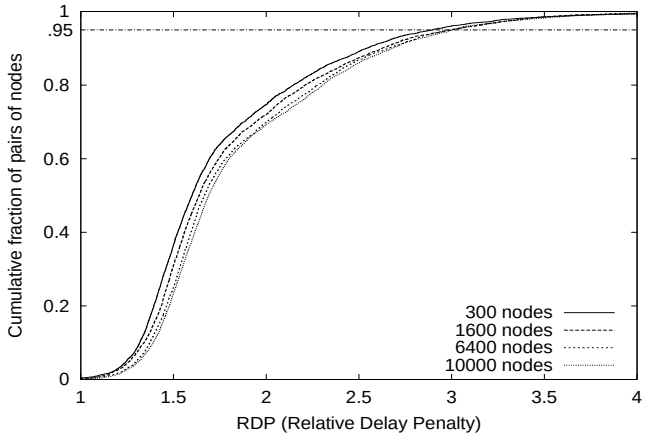

Fig. 6. Cumulative distribution of RDP shown for various overlay sizes.

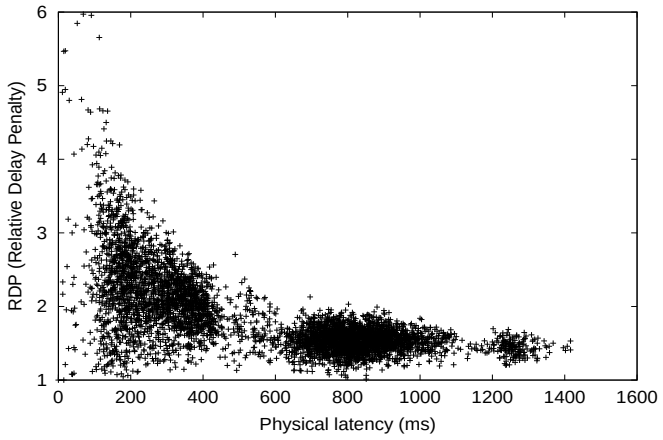

Fig. 7. RDP vs Physical Latency, Overlay size 10,000, ECS = 30

this good performance is that children are very close to their heads, which reduces the overhead for the child in routing through the head.

For flat overlays, [6] observed that higher RDPs are associated with pairs of nodes having small physical latency between them. Figure 7 shows the same behavior for our hierarchical overlays.

B.3 Cluster Size

In this section, we study the impact of cluster size on latency. We varied the expected cluster size for an overlay with 400 nodes. Figure 8 shows that cumulative distribution of RDPs for flat and hierarchical overlays are very similar. To investigate the effect of cluster size on larger overlays, we repeated the same experiment for a 3200-node overlay. Figure 9 shows 95% RDP as we increase the expected cluster size and again we see very little variation.[3]

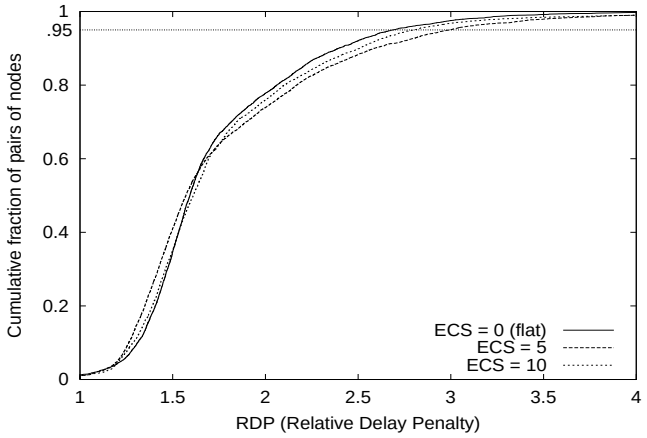

Fig. 8. Cumulative distribution of RDP shown for various expected cluster sizes in a 400 node overlay

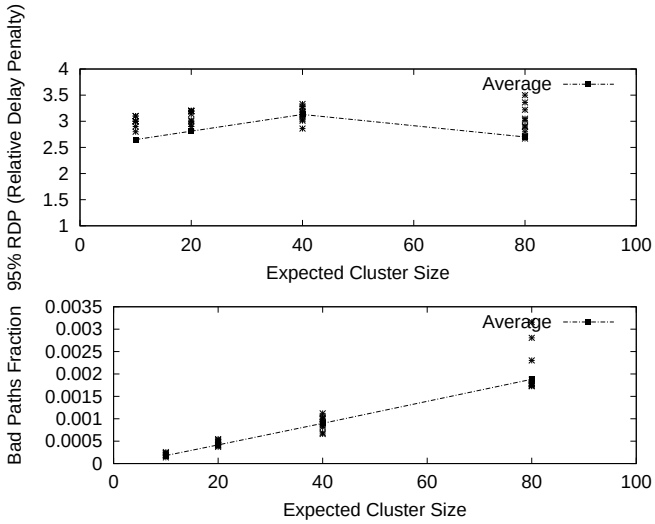

Fig. 9. Effect of (ECS) Expected Cluster Size in a 3200 nodes overlay. The top graph shows the variation of 95% RDP. The bottom graph shows variation of the fraction of bad paths (RDP > 5).

The results above are counter intuitive: one would expect the performance to degrade as bigger clusters are formed because the overhead of going through the head increases. On deeper investigation, we found that degradation in performance can be seen by looking at the number of paths that suffer high RDP. To quantify this we define *Bad Paths Fraction* as the ratio of number of paths with RDP > 5 to total number of paths. The lower graph in Figure 9 shows that the *Bad Paths Fraction* increases with increasing cluster size, but the fraction is too small to be

[3]Ideally, we would like to compare the results with flat overlays of the same size, but we could not simulate flat overlays beyond a few hundred nodes due to CPU limitations.

reflected in the 95% RDP. However, almost all of the "bad paths" are associated with nodes that are physically very close, as shown in Figure 7.

C. Other Effects of Hierarchy

Now we investigate the effect of hierarchy along the following three dimensions, control overhead, time to respond to dynamic conditions, and load balancing in routing.

C.1 Protocol Overhead

A fundamental benefit of hierarchy is that protocol overhead (bandwidth requirements for overlay maintenance) is reduced, because the mesh management protocol is run among a smaller group of nodes. We discuss bandwidth requirements for both head and child nodes in a hierarchical overlay. The protocol overhead can be broken down into the two components: mesh management and clustering.

1. **Mesh management overhead**: Nodes in the mesh have to periodically exchange routing tables with their neighbors (number of neighbors is bounded by a small constant), and other nodes (to evaluate add/delete link operation). The exact number of exchanges is a constant depending on the frequency at which these operations are done. If the mesh management is operating among m nodes, the size of each exchange is $O(m)$ since the routing table contains information about all the nodes. Therefore, bandwidth requirement for any node in the mesh is $c_1 m$, for some constant c_1.

In a hierarchical overlay, mesh management protocol is running both at the top level, and with in each cluster. In an overlay with n nodes and expected cluster size k, number of head nodes are n/k. Overhead due to the top-level mesh management on every head node is $c_1(n/k)$. Overhead due to the mesh management with in one cluster (of size k) on every child node is $c_1 k$. Since head is also operating in the mesh management protocol with in the cluster, overhead on a head node is $c_1(n/k + k)$.

2. **Clustering overhead**: Clustering requires a child to probe its parent and other head nodes in the overlay to evaluate migration. Recall that in the protocol child probe a small constant number of head nodes per unit time. This way every child generates constant number of messages periodically. Since each message is constant size, it causes an overhead of c_2 on child nodes, for some constant c_2 (c_2 is much smaller than c_1). Additionally, each head node has to respond to these probes. Assuming uniform distribution of child pings on head nodes, every head node has to respond to about k pings, which causes an overhead of $c_2 k$ on them.

Therefore, the total overhead on head nodes is $c_1(n/k + k) + c_2 k$, which is $c_1 n/k + (c_1 + c_2)k$. For n much larger than k (like $n = 10,000, k = 20$), the bandwidth overhead reduces roughly by a factor of k. Total overhead on child nodes is $c_1 k + c_2$, which is much less than overhead on head nodes.

Based on the above analysis and the parameters in our simulations, the bandwidth requirement for every node in a flat overlay of 10,000 nodes is about 400 Kbps. On the other hand, for a hierarchical overlay of 10,000 nodes with cluster size 20 and 500 head nodes, overhead is much less. For head nodes its only about 20Kbps and for child nodes its less than 1Kbps. (The numbers for hierarchical overlays have been confirmed using simulation.) Note that most of the nodes in the overlay are child nodes, hence the gain is substantial.

C.2 Responding to Changes

Hierarchy is better suited to frequent changes because of information hiding, as a result of which the effect of a change is limited to only a few nodes. To demonstrate this, we injected events in an overlay of size 200 and observed how much time it took for the overlay to stabilize to a new topology and how many messages it generated in the process. The events induced were 20 nodes leaving, 20 nodes joining, 40 nodes leaving and 40 nodes joining. Enough time was given for the overlay to stabilize between consecutive events. We compare responses of three overlays of size 200 - flat, cluster size of 20, and cluster size of 40 - in Figure 10. The flat overlay responded with much more changes for a much longer period as compared to the hierarchical overlays, and the overlay with bigger cluster size is better than the one with smaller cluster size. For instance, when 20 nodes are brought down, the hierarchical overlays took less than 10 minutes to settle down to the new topology, while the flat overlay took about 40 minutes. Because hierarchical overlays stabilize much faster, they are more suited towards large applications with dynamic conditions.

C.3 Load Balancing

By definition, in a hierarchy all nodes are not equal. In our overlays, management overhead is not shared equally by all nodes. Head nodes have more responsibilities. Section V-C.1 showed how the control bandwidth requirement for the two node types is different. Another dimension of responsibility is routing.

To characterize how evenly the responsibility of unicast routing is distributed among the head nodes, we define *node stress* as the ratio of number of unicast paths that go through this node to the total number of paths (all node-

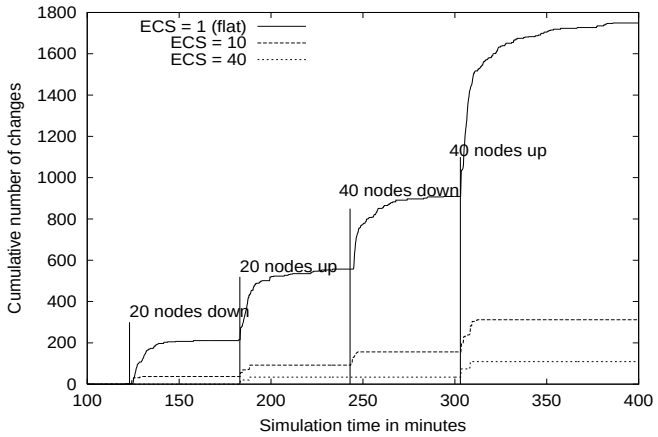

Fig. 10. Changes in the overlay topology (migrate + add link + delete link) vs Simulation time for different cluster sizes

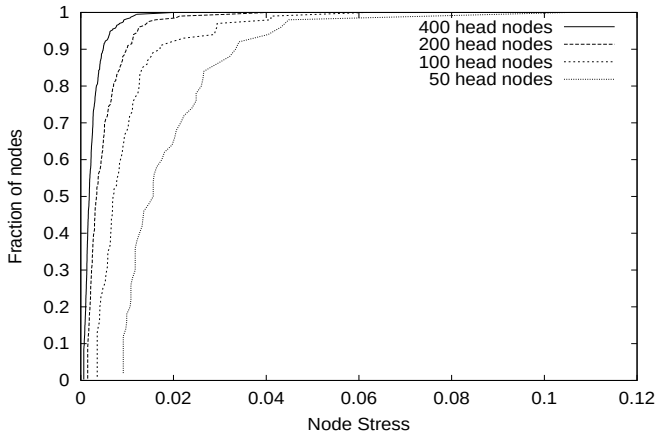

Fig. 11. Node Stress on head nodes for different overlays.

pairs). Child nodes only forward traffic for child nodes in the same cluster, and hence node stress on them would be very low (close to $1/n$, for all paths involving the node itself, because most paths cross cluster boundaries).

Figure 11 plots the node stress on head nodes for different clustering levels in an overlay size of 400. There are two things to be noted in the graph. First, the load distribution among the head nodes themselves is fairly uniform (a quickly rising vertical line in the plot). Second, and more important, stress goes up as the levels of clustering is increased. So the routing load on an individual head node goes up as the number of head nodes is decreased. This provides a case against very big cluster sizes, unless the head nodes have enough resources to deal with the increased routing load presented. As described in Section IV-C.2, in a heterogeneous environment this can be alleviated by picking the right head nodes.

D. Recovery From Failures

The protocol has built-in mechanisms to detect and recover from failures. We quantify the recovery performance in this section. Death of a child node is a simple case to handle. Its effect is localized to the cluster it is in; it does not effect rest of the nodes in the overlay. Failure of head nodes is more critical, as it not only orphans the children, but can also partition the top-level topology. Hence, we restrict our attention to failure of head nodes only. When a head node dies, its children migrate to another head as soon as they detect the head's death. At the same time, any partitions at the top-level are detected and repaired.

The simulation in Figure 12 has 1600 nodes in the overlay, with expected cluster size of 5. A fraction of the head nodes are chosen randomly, and killed. The figure shows the *disconnectivity* in the overlay with time. *Disconnectivity* is defined as the ratio of pair of nodes that cannot reach each other to total pairs. Although the disconnectivity level increases as more head nodes fail, the time taken to recover to a connected topology remains about the same, 3 minutes, which corresponds to the time out constants used in simulation for detecting and repairing partitions.

An overlay robust to failures should have both fast detection and fast adaptation to changes. Figure 12 also plots the number of changes (in 10-second bins) that occur in the overlay topology as a result of induced failure. A change is occurrence of child migration, addition of a new link or deletion of an existing link. This reflects how much time it takes for the protocol to reach a new stable state. From the graph, it is clear that there is a flurry of activity in the first 5 minutes after detection, and then rate of changes comes down significantly. The later events mainly lead to incremental improvements.

From this it is clear that the protocol is able to maintain connectivity even at high failure rates of head nodes. In practice head nodes would be chosen based on their stable participation which would reduce their failure rate and therefore reduces the transient *disconnectivity* in the overlay.

VI. Resource Discovery in the Wide Area

In this section we evaluate the performance of our protocol in context of broadcast based applications like Gnutella[4]. We would like to compare the overlay formed by our protocol with the existing Gnutella overlay. We implemented a Gnutella-like protocol to generate Gnutella overlays. It creates an overlay topology by randomly connecting nodes taking into account a degree distribution found in Gnutella overlays [19]. While this protocol might not yield overlay topologies identical to Gnutella (because

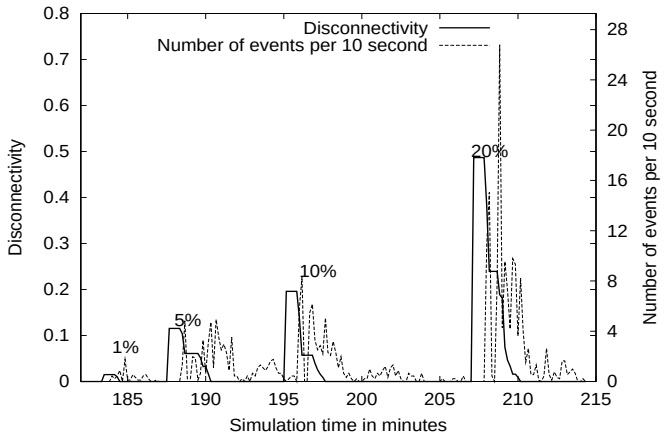

Fig. 12. Disconnectivity and Total number of changes in the overlay topology vs Simulation time. % value denotes % of head nodes failing.

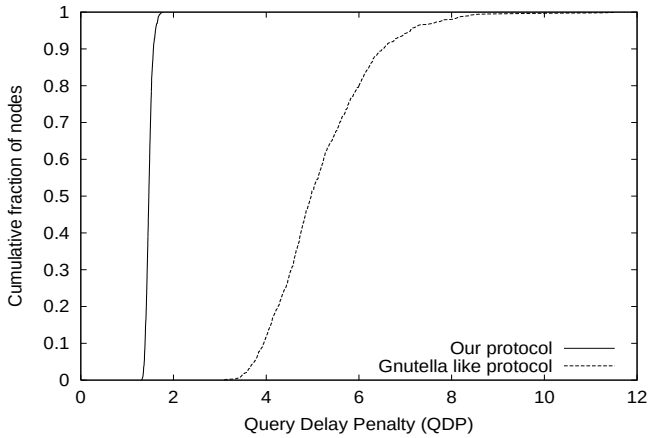

Fig. 13. Comparison of our overlay with a Gnutella-like overlay

of the presence of several degrees of freedom involving humans in the way the Gnutella network works), we believe it provides a reasonable comparison point.

We use two metrics for evaluation: the first measures the latency overhand, and the second overhead on the network.

A. Query Delay Penalty

Query Delay Penalty (QDP) for a node is defined as the ratio of broadcast latency using the overlay to broadcast latency using the physical network. Broadcast latency is the time taken for a query originating from that node to reach all other nodes of overlay, which is the same as the maximum latency from this node to any other node in the overlay. QDP is a measure of latency overhead incurred by a broadcast-based application when using the overlay, just like RDP measures the latency overhead for unicast applications.

Figure 13 compares the QDP for two overlays of 1600 nodes, one formed by our protocol and other using the Gnutella-like protocol. With our protocol QDP observed by all nodes was less than 2. On the other hand, for the Gnutella-like overlays more than 90% of the nodes had QDP more than 4, and the worst case QDP was more than 8.

B. Resource Usage

In broadcast, a message traverses each overlay link once or twice. Assuming cost of traversing a tunnel is proportional to its latency, an overlay with fewer low latency links is more efficient for broadcast purposes than the one with many large latency links. We define *resource usage* as sum of latencies of all links in the overlay. Note that good RDP does not imply low resource usage. For example an overlay in which every node is connected to every other node has RDP 1, but has very high resource usage.

The resource usage of Gnutella-like overlay was more than 10 times that of our overlay. Hence, we are able to achieve better performance using fewer resources.

VII. Conclusions and Future Work

In this paper we have presented a protocol to form self-organizing overlays that scales to at least tens of thousands of nodes. The protocol builds a two-level hierarchy using two techniques: clustering and mesh-management. Clustering builds the hierarchy dynamically and mesh management forms the overlay at each level of hierarchy. Our key conclusions are:

- Hierarchy has minimal effect on performance metric such as Relative Delay Penalty (RDP).
- At the same time, hierarchy reduces the bandwidth requirements of control traffic. Roughly, the gain is a factor of k for head nodes, and a factor of n/k for child nodes, where n is the overlay size, and k is the cluster size.
- Hierarchical overlays absorb changes better than flat overlays. They stabilize more quietly and with fewer changes, and so are well-suited to dynamic environments. The stability of hierarchical overlays can be further increased by exploiting the heterogeneity among overlay nodes.
- For broadcast-based applications, our hierarchical overlays yield major performance benefits (more than a factor of 4 in our simulations) while cutting down on the resource consumption (by a factor of 10) compared to ad hoc techniques like Gnutella.
- Our extensions to Narada lead to significant improvements in RDPs. In our simulations, the 95th percentile RDP was reduced by 25%.

It is important that a self-organizing overlay protocol dynamically adapt to the workload presented to it, to preclude performance bottlenecks (hotspots). While some of the means to alleviate hotspots would be application-specific, there are some measures that can be taken independent of the application being run on top of the overlay. For instance, if some node in the overlay is overwhelmed by traffic it forwards between two neighbors, it could request its neighbors bypass it and to send data directly. Or, if some node observes an unacceptable level of congestion over a tunnel, it could either find an alternate tunnel to destination or divide the traffic that goes over that tunnel. As part of future work, we intend to investigate application-independent techniques to address the problem of hotspots in overlays.

Large-scale overlays consisting of end-hosts in the Internet have to be able to operate under highly dynamic environment. We intend to measure the performance of our overlay protocol under such conditions. This requires modeling the life time distributions of the overlay nodes and other dynamic conditions like physical link failures. On a related note, we also plan to investigate how hierarchy can be used to exploit heterogeneity among the overlay participants.

The protocol presented in the paper, creates a two level hierarchy. Creating multiple levels of hierarchy to achieve further scalability is another interesting future direction.

ACKNOWLEDGEMENTS

We are grateful to Andy Collins and Andrew Whitaker for extensive discussions during the design phase of the project. Scott Shenker and Srinivasan Seshan provided very useful feedback. We are also thankful to Stefan Saroiu and Neil Spring for reviewing a draft version of this paper.

REFERENCES

[1] H. Eriksson, "MBone: The Multicast Backbone," in *Communications of the ACM*, August 1994, vol. 37, pp. 54–60.

[2] "ABone Web Pages," http://www.isi.edu/abone/.

[3] "6Bone Web Pages," http://www.6bone.net/.

[4] "Gnutella Web Pages," http://www.gnutella.com.

[5] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," Tech. Rep. 01-06-02, UW-CSE, June 2001.

[6] Yang-hua Chu, Sanjay Rao, and Hui Zhang, "A Case for End System Multicast," in *ACM SIGMETRICS*, June 2000.

[7] Yatin Chawathe, S. McCanne, and Eric Brewer, "An Architecture for Internet Content Distribution as an Infrastructure Service," February 2000.

[8] John Jannoti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," in *OSDI*, 2000.

[9] Paul Francis, "Yoid: Your Own Internet Distribution," http://www.aciri.org/yoid/.

[10] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicasting Routing Protocol," RFC 1075, IETF, 1988.

[11] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley, "The Design and Implementation of an Intentional Naming System," in *SOSP*, December 1999.

[12] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A Scalable Content Addressable Network," in *ACM SIGCOMM*, September 2001.

[13] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, September 2001.

[14] Anthony Rowstron and Peter 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)*, November 2001.

[15] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep. UCB/CSD-01-1141, UCB, April 2001.

[16] Kortsarz and Peleg, "Generating low-degree 2-spanners," in *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[17] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan, "Detour: a Case for Informed Internet Routing and Transport," in *IEEE Micro*, January 1999, vol. 19, pp. 50–59.

[18] "GT-ITM:Modeling Topology of Large Internetworks," http://www.cc.gatech.edu/projects/gtitm/.

[19] DSS2 Clip System, "Gnutella: To the Bandwidth Barrier and Beyond," "http://dss.clip2.com/gnutella.html".