ABSTRACT

Warrier, Ajit C. Proximity Induced Labeling Schemes for Distributed Hash Tables. (Under the direction of Dr. Injong Rhee and Dr. Khaled Harfoush).

P2P systems have been recently introduced as an unconventional approach to networking. Among them, structured P2P systems (or Distributed Hash Tables) have such benefits as load balancing, scalability, and self-organizing nature. Most of the earliest structured P2P systems had virtualized address spaces, hence disregarding underlying physical topologies while creating the overlay. By incorporating knowledge of the underlying topology into the P2P system, efficient overlays can be constructed. There have been several different approaches towards this goal. The most popular approach has been reactive in nature, where nodes having been assigned their virtual identifiers in the overlay, search for good neighbors or routes towards their destination.

This work, on the other hand, takes a proactive approach. Our goal is to assign identifiers to nodes so that their position in the overlay would approximately reflect their position in the physical topology. Such identifiers or *Proximity Induced Labels* would then make the consequent search for good neighbors/routes unnecessary, since they would be implicit by the overlay geometry. We introduce two such labeling techniques, one for the well known Content Addressable Network (CAN), and the other for the binary Hypercube, based on delay information from a set of well-known nodes on the Internet called *Landmarks*. Our performance evaluation demonstrates that proximity induced labels can be assigned in a scalable manner to CAN without changing the CAN algorithms, leading to better performance than the conventional CAN. Also, such labeling when combined with the high connectivity of the Hypercube, achieves highly efficient overlays at the cost of some increased node state.

Proximity Induced Labeling Schemes for Distributed Hash Tables

by

Ajit Chakrapani Warrier

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2004

Approved By:

Dr. Khaled Harfoush	Dr. Jaewoo Kang

Dr. Injong Rhee
Chair of Advisory Committee

Dedicated to my family, for their love and unwavering faith and support.

Biography

Ajit Chakrapani Warrier was born on July 7, 1980 in Jabalpur, India. He graduated with a Bachelor of Engineering degree in Computer Engineering from Nirma Institute of Technology, Gujarat University, India in June 2002. He worked with Tata Consultancy Services, Ahmedabad, India as part of his final semester project. He then joined the Masters program in Computer Science at North Carolina State University. After graduation, he plans to continue at NCSU as a Ph.D. student in the Computer Science program.

Acknowledgements

I would like to thank my advisors Dr. Injong Rhee and Dr. Khaled Harfoush for being a constant source of inspiration and insight. Their valuable ideas have played an important and indispensable role in the making of this thesis. I also thank Dr. Jaewoo Kang for being on my advisory committee and providing valuable comments.

I thank my friends and roommates Kunjal Shah, Anubhav Dhoot and Ajit Rajagopalan for some great times as a graduate student. Life at NC State would not have been the same without them.

Contents

List of Figures

Chapter 1

Introduction

Peer-to-Peer (P2P) systems have been recently put forward as an unconventional approach to computer networking. The flexibility and robustness of structured P2P systems (also known as Distributed Hash Tables or DHTs) like CAN [17], Chord [22], Pastry [20] and Tapestry [29] have been demonstrated in recent papers. In the pure form, none of these overlay structures have information about the underlying physical topology. The ability to incorporate physical topology information into P2P systems would be crucial during the real-world deployment of these systems in various areas such as file sharing, application layer multicast and content distribution networks. As research in P2P systems has matured over the years, attempts have been made to include information about the physical topology into the overlay geometry. These attempts can be classified into three different categories, as mentioned in [9]:

1. **Proximity Neighbor Selection (PNS):** The neighbors in the routing table of the P2P node are chosen based on their proximity.

2. **Proximity Route Selection (PRS):** Once the routing table is chosen, the choice of the next-hop when routing to a particular destination depends on the proximity of the neighbors.

3. **Proximity Induced Labeling (PIL):** Node identifiers are chosen based on their geographic location.

DHTs have varying degrees of flexibility in terms of neighbor selection, route selection, or node identifier selection, which decides whether they are capable of PNS, PRS or PIL respectively.

DHTs that have the flexibility in neighbor selection essentially select neighbors from a subset of the identifier space. The challenge lies in finding the "closest" such nodes from such subsets. DHTs using PRS take advantage of their flexibility in selecting among multiple routes to the destination. Here the challenge lies in "filtering" the set of possible next-hop neighbors using a heuristic based on proximity. DHTs using PIL takes advantage of the fact that the network topology information is encoded into their identifiers and hence their geometry.

1.1 Motivation

Our goal in this work is to introduce PIL properties in P2P systems. Instead of using Internet positioning techniques like IDMaps [8], GNP [15] and Lighthouses [16] which require extra processing, we aim to do this by simply using latency measurements to a set of well-known nodes on the Internet called *landmarks*. These landmarks may be dumb nodes and do not require any additional responsibility other than being able to respond to ICMP echo messages (pings). We aim to demonstrate that such a P2P system will closely approximate the actual network topology thereby providing latencies close to the actual IP latency. At the same time we need to maintain the scalability and robustness that makes P2P systems so powerful.

1.2 Contribution

We present two techniques; the first one incorporates PIL into an already existing DHT, CAN [17]. This scheme requires no change to existing CAN algorithms for node join/leave. We compare the performance of this scheme with the CAN Binning scheme [18], which is another PIL scheme for the CAN overlay. We show that the performance of CAN Binning depends on the number of landmarks being used and the underlying topology of the overlay. Also, the use of CAN Binning destroys the uniform load-balancing properties of CAN. This makes it unsuitable for the dynamic Internet. From our simulations, we find that although CAN Binning performs better, our scheme does not suffer from this unpredictability and our performance never degrades with the use of more landmarks.

The second technique presents a PIL scheme for the Hypercube overlay. Our simulations show that the combination of the high connectivity of Hypercubes and the proximity induced labeling of nodes allows the construction of highly efficient overlays at the expense of some increased node state.

1.3 Organization

The rest of the thesis is structured as follows. Chapter 2 presents the related work in the area of topology aware overlays. In Chapter 3, the Binning scheme for CAN is introduced. In Chapter 4, our scheme for PIL of nodes in CAN is presented. In Chapter 5, we introduce the simulation setup and metrics we use for our performance results. In Chapter 6, we present the scheme for PIL of nodes in the Hypercube overlay and the corresponding simulation results in Chapter 7. Finally, we conclude in Chapter 8, with a summary of findings and future research in this area.

Chapter 2

Related Work

DHT systems first came into prominence with the introduction of two systems CAN [17] and Chord [22]. DHT systems distribute the load among peers equally by the virtualization of the address space. The first work which puts forward the suggestion that virtualization is *not* a good idea is [11]. The authors argue that virtualization destroys locality and give an example of TerraDir [21], a non-virtualized overlay in the form of a rooted hierarchy which explicitly codifies the application hierarchy. Our goal in this work is to achieve locality within the limits of DHT, and hence alleviating the drawbacks of DHT.

In CAN, the nodes are mapped onto a d-dimensional Cartesian space. Hence the fundamental geometry resembles that of a Hypercube. Thus CAN nodes do not have the freedom to select their neighbors, the neighbors are implicit in the labeling of the node. On the other hand, the CAN structure enjoys the freedom of having multiple possible routes to a destination. This allows CAN to use PRS but disallows the use of PNS. The way PRS is implemented in CAN is that among the possible next-hop neighbors, messages are forwarded to that neighbor which has the best ratio of progress in the Cartesian distance to network delay cost. [18] also proposes a PIL solution over the CAN system using the binning scheme which is explained in detail in later sections.

E-CAN or Expressway CAN [26] augments the basic CAN structure with routing tables of larger span. Each node not only knows about its immediate neighbors, but also maintains neighbour information about high-order zones, which encompass many CAN zones. Using this information, instead of going through each intermediate node in CAN, messages pass through in hops which stride through several zones, hence reducing the end-to-end delay.

In Chord, the underlying geometry is a ring structure. Each node a in a system of n nodes, maintains $\log n$ neighbors, called *fingers*, where the i^{th} neighbor is the node closest to $a + 2^i$ on the ring. Although not a feature of ring topologies in general, the Chord architecture defines a specific set of neighbors. Thus in its pure form, Chord is incapable of PNS. Also the optimal path in terms of hops in Chord is O($\log n$). But sub-optimal paths traversing more nodes may be used (but with less latency), and this makes it suitable for PRS.

In Kademlia [14], the underlying geometry is the XOR-network. Nodes use 160-bit identifiers as their labels. These labels are modeled to be the leaves of a binary tree. The distance between two nodes is the exclusive OR (XOR) of their labels. Each node maintains $\log n$ neighbors, where the i^{th} neighbor is any node within an XOR distance of $[2^i, 2^{i+1}]$ from itself. The XOR network is very flexible in that it allows a wide selection of neighbors, and also allows multiple routes to the destination. Thus it is well-suited for both PNS and PRS.

Pastry [20] follows a "hybrid" approach, where both tree and ring geometries are employed. Nodes take their labels from a circular node ID space, and a label is a sequence of digits with base 2^b. The routing table of a node consists of up to $log_{2^b}N$ rows with $2^b - 1$ entries. Each of the $2^b - 1$ entries at the n^{th} row of the routing table refers to a node whose label shares with the present node's label the first n digits, but whose $(n+1)^{th}$ digit has one of the $2^b - 1$ possible values other than the $(n+1)^{th}$ digit in the present node's label. Routing is achieved by means of prefix matching. Pastry maintains a hybrid network of tree and ring, where each node is located at

the leaf of the tree and also a point in the ring. For the n^{th} row of its routing table, a node has the freedom to select "close" neighbors among those who share the first n digits in their label. Each hop made using tree geometry causes the distance to the destination to reduce by one digit, thereby increasing the shared number of digits with the destination label. On the other hand, hops made using ring geometry bring the message numerically closer to the destination, keeping the number of shared digits the same. Optimally, Pastry could reach its destination using just tree hops. But sub-optimal routes using both tree and ring hops could lead to less latency. Thus Pastry allows for both PNS and PRS.

Tapestry [29] is similar to Pastry in that it uses a prefix matching algorithm to route queries, but it also includes replication on the nodes for added robustness and performance.

Viceroy [13] is a novel P2P structure based on a butterfly network with a constant amount of state information and O(log n) routing complexity. The constant state information results in it being unsuitable for either PRS or PNS.

Mithos [24] uses a mesh network. New nodes joining the overlay network incrementally find better neighbors by using delay measurements. Once it has found the closest node to itself, it uses the coordinate of the neighbors to determine its coordinate. The problem with this approach is that it may require each node to perform an excessive number of delay measurements to find its closest peer node.

Another approach that could be taken in the case of a mesh structured overlay is to develop some kind of a model of the Internet using Cartesian coordinate space. Tools like IDMaps [8], GNP [15] and Lighthouses [16] are proposed for the problem of finding the position of a host relative to other hosts on the Internet. IDMaps requires the distribution of machines around the Internet so that every address-prefix is close to one of them, while GNP assigns the coordinates of hosts based on their distance to a set of passive landmark hosts. Lighthouses adds more flexibility to GNP by allowing a host to find its coordinate based on delays to only a subset of the complete

landmark set.

In [2], Banerjee et al. also propose a nearest neighbor algorithm for positioning overlay nodes. They assume, however, that the P2P overlay is organized in a hierarchical fashion. At the lowest hierarchies, the nodes are clustered according to their topological proximity.

One interesting implication of topology aware DHTs is the idea of applying them to Application Layer Multicast(ALM) [5]. DHTs with their fault tolerant, self-organizing nature coupled with topology awareness have been seen as a viable framework for ALM. ALM has been implemented over CAN [19], Pastry [3], and Tapestry [30].

Among the various ALM techniques, one of particular interest to us is Hypercast[12]. In Hypercast nodes have binary labels and they arrange themselves into a logical Hypercube. The labels are assigned in gray order, and hence have no relation to their physical topology. The PIL solution in this work is also implemented over the Hypercube, but the node labels reflect the position of the node in the physical topology. Also, for the purposes of gray order label assignment, Hypercast imposes the limitation that nodes have to join one by one and that when a new node enters the overlay, it needs to inform all other nodes in the overlay about it. These two limitations make it unscalable for use in overlays with large member sets. Our Hypercube overlay does not face either of these limitations.

Chapter 3

The Binning Scheme

3.1 Introduction

The fundamental goal of the the *Binning scheme* is to provide for congruence between the physical topology and the P2P overlay network. This is accomplished by partitioning nodes into "bins" such that nodes within a bin are relatively closer to one another than to nodes in a different bin. For the purpose of binning, the availability of a well-known set of machines that act as landmarks on the Internet is assumed.

A node that wishes to join the overlay would first measure its round-trip-time (RTT) to each of these landmarks and orders the landmark IDs in the non-decreasing order of RTT. This ordering represents the bin that the node belongs to. Physically close nodes are likely to have the same ordering and hence will belong to the same bin. With n landmarks we have $n!$ different orderings possible, and hence $n!$ different bins. It is important to note that many of these bins represent orderings which may not be actually possible in the underlying topology, and hence no nodes will ever be hashed into such bins.

In [18], the authors propose a trivial embedding of these bins, which resulted in a random placement of these bins. We next describe this embedding for the CAN P2P overlay.

3.2 The CAN Binning Scheme [BCAN]

In CAN [17], nodes are mapped to a virtual d-dimensional Cartesian coordinate space. Every node has an average of $2d$ neighbors in the overlay. Since the nodes are randomly mapped into the Cartesian space, the overlay structure has no resemblance to the underlying physical topology. For a system with n landmarks, this Cartesian space is divided into $n!$ equally sized portions. Each of these $n!$ portions corresponds to a landmark ordering as described in the section above. This is done in the following manner: assuming a fixed cyclical ordering of the dimensions (e.g. xyzxyz...), we first divide the space, along the first dimension, into n portions, each portion is then sub-divided along the second dimension into $n-1$ portions each of which is further divided into $n-2$ portions and so on. Now, instead of randomly mapping a node into the Cartesian space, a CAN node must first find its bin, based on its delay measurements to the landmarks. The new node then joins the CAN at a random point in that portion of the coordinate space associated with its bin.

Using this binning scheme, a node newly joining the CAN overlay can peer with the set of nodes which are physically close to themselves. But the embedding of the bins into the CAN as presented in [18] presents some new problems of its own:

1. The binning system does not maintain any order between bins. Thus although the latency between nodes within a bin is low, inter-bin latencies could be arbitrarily high.

2. As the number of landmarks is increased, the number of possible bins also increases. This improves the performance due to the reduced latency between nodes in the bins by providing more selectivity. But on the other hand, further increase in the number of landmarks will reduce performance. This is because there will be less nodes per bin, and most of the queries will be between inter-bin nodes. The optimal number of landmarks depends on the underlying physical topology which makes it difficult to predict accurately.

3. When we use the binning scheme in the CAN overlay, the hash space is non-uniformly divided among the nodes. It is well known that this causes problems with load-balancing as some nodes are responsible for more content, which goes against the P2P paradigm where all nodes are equally responsible.

In later sections we present two new binning schemes which address the above mentioned problems of binning. We use the simplicity of binning to provide labels to nodes which approximate their location relative to each other in the underlying physical topology.

Chapter 4

The Laminar CAN Binning Scheme [LCAN]

4.1 Introduction

In this section, we describe an overlay organization scheme designed to solve the deficiencies in the original binning scheme mentioned in the previous section. Our proposal is as simple and efficient to deploy as the original scheme. The central idea is to map parts of the CAN hash space (zones) to landmarks, maintaining the topological relationship between them. Nodes find the closest landmark by means of measuring the RTT to the set of landmarks. A random point within the zone alloted to that landmark is the initial id of the node. As more nodes join the system, some zones become more populated than others. To maintain the uniformity of the CAN space, we dynamically change the zone sizes. The boot strap server needs to maintain the population of nodes mapped to each of the zones.

4.2 Laminar Binning

Our proposed scheme requires extra tolerable complexity at the bootstrap server and landmarks that measure the latency between themselves. These landmarks are similar to tracers in IDMaps [8] that are already deployed and the bootstrap server could be a HOPS server, which has the distance information between the tracers and can identify the Address Prefix (AP) to which a node belongs and the closest tracer to this AP. The method has two components:

1. Embedding the landmarks onto a 2-dimensional CAN space while maintaining appropriate distance (latency) relations between the landmarks. One bin (the area of the hash space occupied by the bin) corresponds to one landmark and this bin is designed to accommodate nodes that are physically close to the corresponding landmark.

2. Maintaining a uniform distribution of nodes in the CAN space in order to alleviate the problems associated with the discrepancy in the zone sizes.

We next elaborate on how to achieve the above goals through an example. Consider a set of five landmarks $L = \{L_1, ..., L_5\}$ and the distances (latencies) along the paths connecting them as shown in Figure 4.1(A).

4.2.1 Landmark Mapping Algorithm

Our approach to organizing landmarks in the CAN overlay begins by transforming the *graph* of landmarks into a *tree* in which nodes represent *laminar* subsets [6] of landmarks.

A **laminar family** $F \subseteq 2^L$ is a family of subsets of L such that for any $A, B \in F$, it is the case that $A \subseteq B$ or $B \subseteq A$ or $A \cap B = \phi$.

We use Laminar subsets of L to cluster the landmarks according to their physical proximity. The closer landmarks are to each other, the more they appear in the same

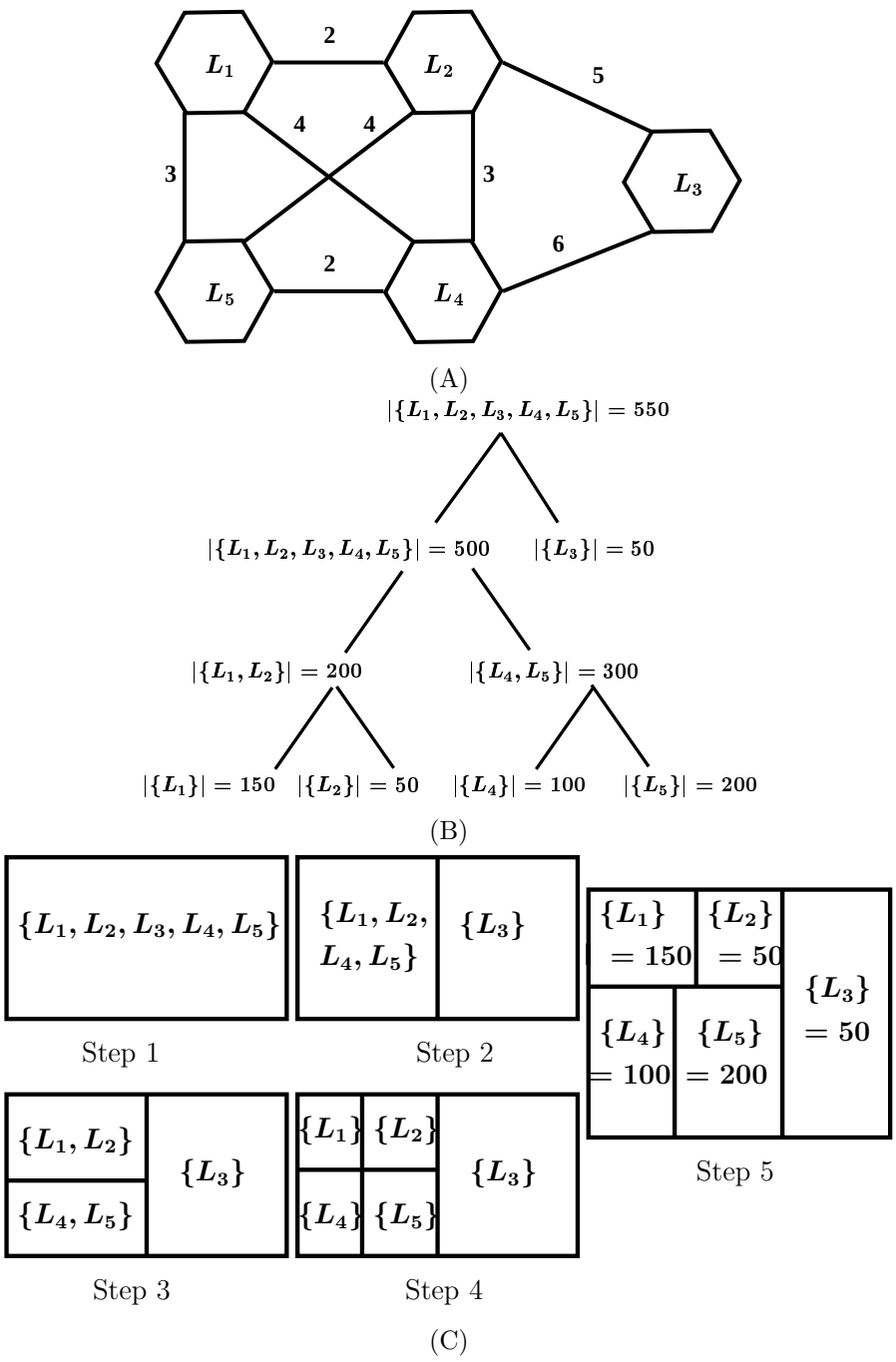


Figure 4.1: Laminar example.

subsets, and the more common ancestors they have in the resulting tree. Algorithm 1 gives the pseudo-code for the creation of such a tree of laminar subsets.

Algorithm 1, starts with a set D_1, then progressively, at each level i, divides the previously obtained sets at level $i-1$ into subsets including landmarks that are within β_i latency from each other, where β is initialized to be the landmark graph diameter and decreases exponentially at each level. The network diameter for the graph in Figure 4.1(A), is 7, yielding $\delta = 3$ and $\beta = 4$. The node L_1 finds nodes L_2, L_4, L_5 within this distance to itself. The only unassigned node L_3 creates a subset for itself at this level and hence set $D_2 = \{\{L_1, L_2, L_4, L_5\}, \{L_3\}\}$. In the next iteration, $\delta = 2$ and $\beta = 2$. Node L_1 finds node L_2 and node L_4 finds node L_5 at this distance from itself, and hence set $D_3 = \{\{L_1, L_2\}, \{L_4, L_5\}, \{L_3\}\}$. At the last iteration, $delta = 1$ and $\beta = 1$ and hence all sets resolve into singleton subsets, and the algorithm terminates. The laminar subsets created in this way are: $D_1 = \{\{L_1, L_2, L_3, L_4, L_5\}\}$, $D_2 = \{\{L_1, L_2, L_4, L_5\}, \{L_3\}\}$, $D_3 = \{\{L_1, L_2\}, \{L_4, L_5\}\}$, and $D_4 = \{\{L_1\}, \{L_2\}, \{L_4\}, \{L_5\}\}$.

The obtained laminar subsets could be inserted in a tree as shown in Figure 4.1(B) with subsets at level i representing nodes at height i of this tree. This tree is used to position landmarks in the 2-dimensional CAN overlay. The pseudo-code for the mapping is given in Algorithm 2. Note that the algorithm also fixes the zone corresponding to each landmark as the population in each zone changes with time. This part of the algorithm is explained in the next section.

As detailed in Algorithm 2, the root node initially occupies the whole CAN space, then nodes are inserted in breadth-first order with sibling nodes splitting the area allocated to their parent in the tree. We split areas along the CAN overlay dimensions interchangeably to avoid creating *thin* zones. Going back to the example, we illustrate how to map our set L of landmarks to the 2-dimensional CAN space, with the steps illustrated in Figure 4.1(C). In Step 1, set L will occupies the whole CAN space. In step 2, we insert sets at level 2. The set $\{L_1, L_2, L_4, L_5\}$ will occupy the position of

its parent. The next sibling, $\{L_3\}$, will try to split its parent, but finds it occupied by the set $\{L_1, L_2, L_4, L_5\}$. Hence each set will occupy half of the CAN space. In Step 3, we insert sets at level 3. It is clear that continuing in this way we achieve the mapping seen in Step 4.

This mapping has the property that landmarks that are physically close, will be close to each other in the CAN overlay; but, they may be close to other landmarks too. For example, landmarks L_3 and L_1 are neighbors in the CAN overlay, although they are not physically close.

Data : Graph connecting a set $L = \{L_1, ..., L_l\}$ of landmarks
Result: Laminar subsets D_i
$Diam$ = Diameter of the landmark graph;
$D_1 = \{L\}$;
$\delta = log_2(Diam)$;
$i = 1$;
while D_i *has non-singleton subsets* **do**
    $\beta_i = 2^{\delta - i} * \beta$;
    $D_{i+1} = \emptyset$;
    for $v = 1$ *to* l **do**
        for *every non-singleton subset* S *in* D_i **do**
            Create a new subset S_v consisting of all unassigned landmarks in S
            within a distance of β_i or less from L_v;
            $D_{i+1} = D_{i+1} \cup \{S_v\}$;
        end
    end
    $i = i + 1$;
end

Algorithm 1: Converting a landmark graph into Laminar subsets.

4.2.2 Maintaining Uniform Distribution of Zone Sizes in the CAN Space

Recall that nodes will be hashed to an arbitrary point in the zone assigned to the landmark they are closest to. As nodes join the system, some bins may become more crowded than others. In order to avoid this problem, we adapt the bin sizes in proportion to the number of nodes assigned to the bins (bin popularity). The bootstrap server(s) keeps count of the number of joining nodes closest to each landmark and use this information to identify the appropriate bin sizes by running Algorithm 2. Figure 4.1(B) provides an example in which 550 nodes are in the system when 150, 50, 50, 100, and 200 nodes are closest to landmarks L_1, L_2, L_3, L_4, and L_5 respectively, and Figure 4.1(C) the resulting area assignment reflecting the difference in landmark popularity.

There could be many bootstrap servers to balance the load of node join and for robustness in the face of server failure. Such servers would need to share node population information so that zone sizes are correctly maintained. Over time some nodes may leave the system and as a result the bootstrap server's count will not reflect the exact number of nodes in the system. This should be fine as long as the node lifetime in all bins is roughly the same; that is, when nodes close to certain landmarks do not leave at a slower or faster pace than nodes close to other landmarks, which is typically the case. Also, this scheme does not enforce strict boundaries between nodes in each bin. This is especially true when a bin size changes as nodes on the border of contiguous bins will not fall into the appropriate bins as perceived by the bootstrap server. As nodes leave the system, nodes that are misplaced will eventually leave the system. Our system is also robust to landmark failures. The loss of a landmark results in its zone being merged with its nearest zone. This results in loss of selectivity, but the system does not fail.

Data : Tree T of the laminar subsets D_i corresponding to the set $L = \{L_1, ..., L_l\}$ of landmarks
Result: An assignment of the landmarks in a 2-dimensional CAN space with zone sizes reflecting node population
Insert T in breadth-first order in a queue Q;
while *BFS queue is not empty* **do**
    n = pop a node from Q;
    if *n is the root* **then**
        Allocate the whole hash space to n;
    else
        $ratio = population(n)/population(parent(n))$;
        if *n is at an even depth in T* **then**
            $splitDim = X$;
        else
            $splitDim = Y$;
        end
        Split dimension $splitDim$ in $parent(n)$ according to $ratio$;
    end
end

Algorithm 2: An allocation of the landmarks in the CAN space while fixing zone sizes based on landmark popularity.

Chapter 5

Performance Comparison Between LCAN and BCAN

5.1 Simulation Setup

The most popular means of generating the topology for P2P systems has been the GT-ITM [27] topology generator. It generates the router level topology of the Internet using a hierarchical approach. Although useful for generating small networks, the topologies generated using GT-ITM do not show the power-law relationships existing at the AS-level and the router-level [7]. Recent studies [9] comparing the behavior of GT-ITM models with real world networks in terms of latency distributions have also showed that the standard practice of applying latencies of 100-20-5ms for Transit-Transit, Transit-Stub and Stub-Stub links respectively may not be correct. This brings studies of proximity methods on DHTs to an unfortunate situation: although we have a correct model for the connectivity of inter networks, we have no provably correct method to label the latencies on the links.

We approach this problem in the following manner:

1. **Topology Generation:** We have used the Inet3 [25] topology generator to

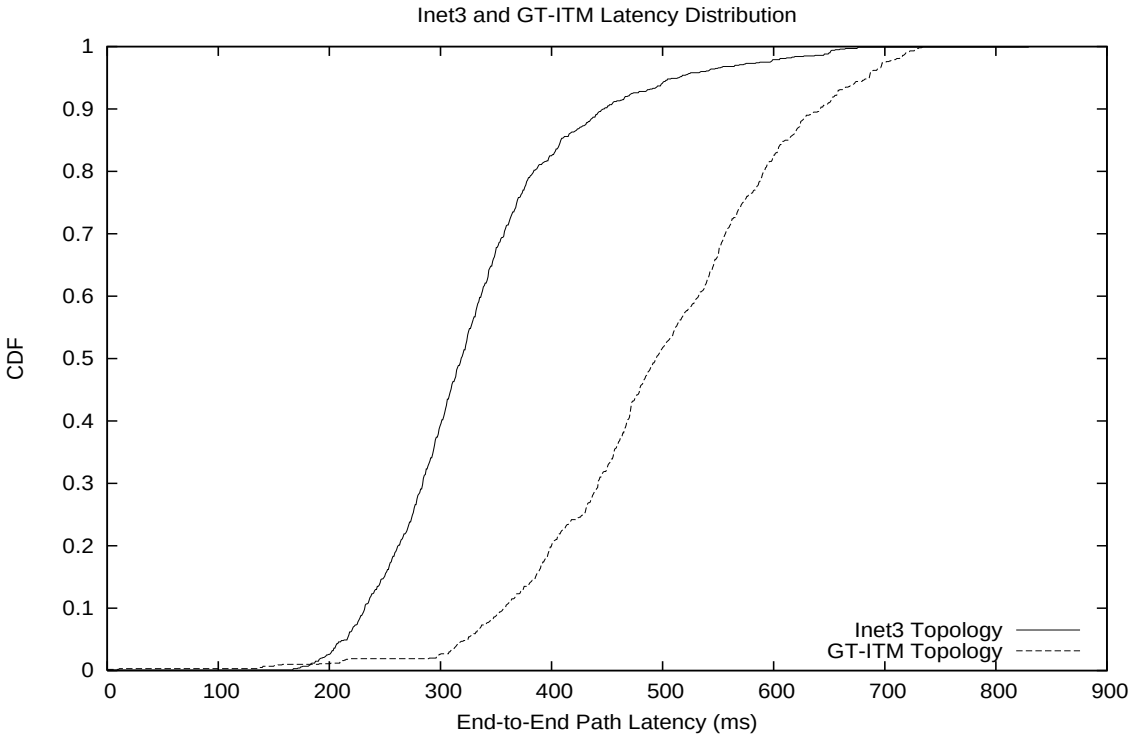


Figure 5.1: Comparison between Inet3 and GT-ITM latency distributions

generate the AS-level topology. This topology generator generates the AS-level connectivity of the network which displays some of the properties of the Internet namely, power-law connectivity [7] and exponential AS growth rate [25]. Due to resource constraints, we use a topology of 3500 ASs.

2. **Labeling of Intra-AS Latencies:** Once we generate the AS-level topology using Inet3, we expand each AS with a mesh, which models the Intra-AS network. ISPs do not usually present information about latency distributions within their ASs but they do give the minimum, maximum and average latencies within their ASs. Most of the top ASs seem to have intra-AS latencies in the range of 10-80ms, with an average of around 40ms. Considering this data we model the intra-AS latency distribution as a uniform distribution from 10-80ms.

3. **Labeling of Inter-AS Latencies:** Here again we are plagued by lack of

experimental data. The general assumption has been that inter-AS latencies comprise the major component of the the end-to-end latencies. A recent study on the ASs controlled by Sprint [28], seems to disprove this long-held belief. In 80% of the studied cases, the inter-AS component turns out to be less than the intra-AS component of the total end-to-end delay. In the remaining cases, propagation delay was found to be the reason for the large inter-AS component. This leads us to believe that on the current Internet, most of the inter-AS latencies are less than or equal to the intra-AS latencies, while a small fraction of the links being "long hop" or inter-continental links, have a much higher latency. This leads us to label the inter-AS latencies as follows: we use the same latency distribution (10-80ms) for 90% of the inter-AS links, while for the remaining 10% of the links, we assign a high latency of 250ms.

For a comparison of the effect of the underlying topology on the performance, we also present results for a GT-ITM topology. This topology has 50 Transit Domains, 10 Stub Domains/Transit Domain, and 100 routers/Stub Domain. Transit-Transit latencies are drawn from a uniform distribution of 20-70ms, Transit-Stub latencies from 2-20ms and Stub-Stub as well as Stub-Router latencies from 0-2ms. Remember that this is in contradiction to the labeling used in the Inet3 topology described above, where the top-level (inter-AS) links have about the same latency as the lower-level (intra-AS) links, with some high delay links corresponding to the inter-continental links. To see the effect of the topology on the latency distribution seen by individual nodes, we present Figure 5.1. The latencies of Inet3 have been scaled down to be compared to that of GT-ITM. The figure shows that Inet3 latencies rise faster, and have somewhat more tail than the GT-ITM latencies. The heavy-tailed distribution of latencies seen by nodes on the Internet has been recently reported in [9]. This gives us some confidence in the latency distribution used in the Inet3 topology.

We do not claim that the Inet3 labeling will present an accurate picture of the current Internet. Indeed without any available data on link latency distributions on

the Internet, it is difficult to prove/disprove such claims. But we do believe that it is much better than the arbitrary labeling used in simulation results presented in previous papers. For consistency we have maintained the same setup for later sections too.

5.2 Performance Metrics

We now look at a comparison between the LCAN and BCAN schemes for different number of nodes in the overlay. As a base case, we present the performance results for the Randomized CAN (RCAN) so that we understand the amount of improvement created by both the techniques. Both LCAN and BCAN are based on the 2-dimensional CAN DHT, and both use landmark delay measurements for their PILs. BCAN uses the full information about their landmarks whereas LCAN uses only the information about the closest landmark. BCAN has been implemented as described in Section 3.2 and LCAN as described in Section 4.2. The three performance metrics are considered:

1. **Latency Stretch** The latency stretch of the overlay is defined to be the ratio of the average inter-node latency on the overlay network to the average inter-node latency on the underlying IP level network. The lower the latency stretch, the better the performance of the overlay. It is a measure of how closely the node organization in the overlay matches the actual physical topology. The use of latency stretch instead of an absolute quantity like the latency allows us to compare the performance across physical topologies.

2. **Variation of Latency Stretch with Number of Landmarks** Since both LCAN and BCAN depend on landmarks for the labeling of nodes, we look at how the number of of landmarks affects their performance.

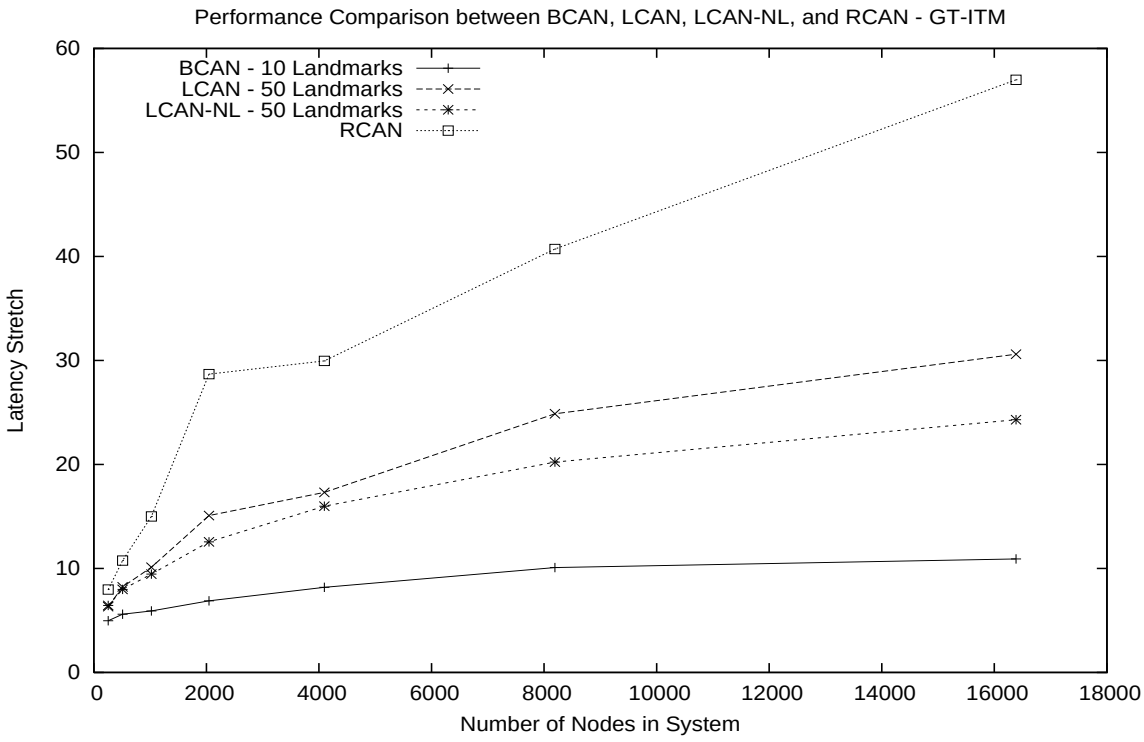
3. **Node Degree Distribution** The node degree distribution tells us how the changes in the CAN overlay have affected its degree distribution. It is a measure of the state information that each nodes has to maintain to participate in the topology aware CAN.

5.3 Latency Stretch

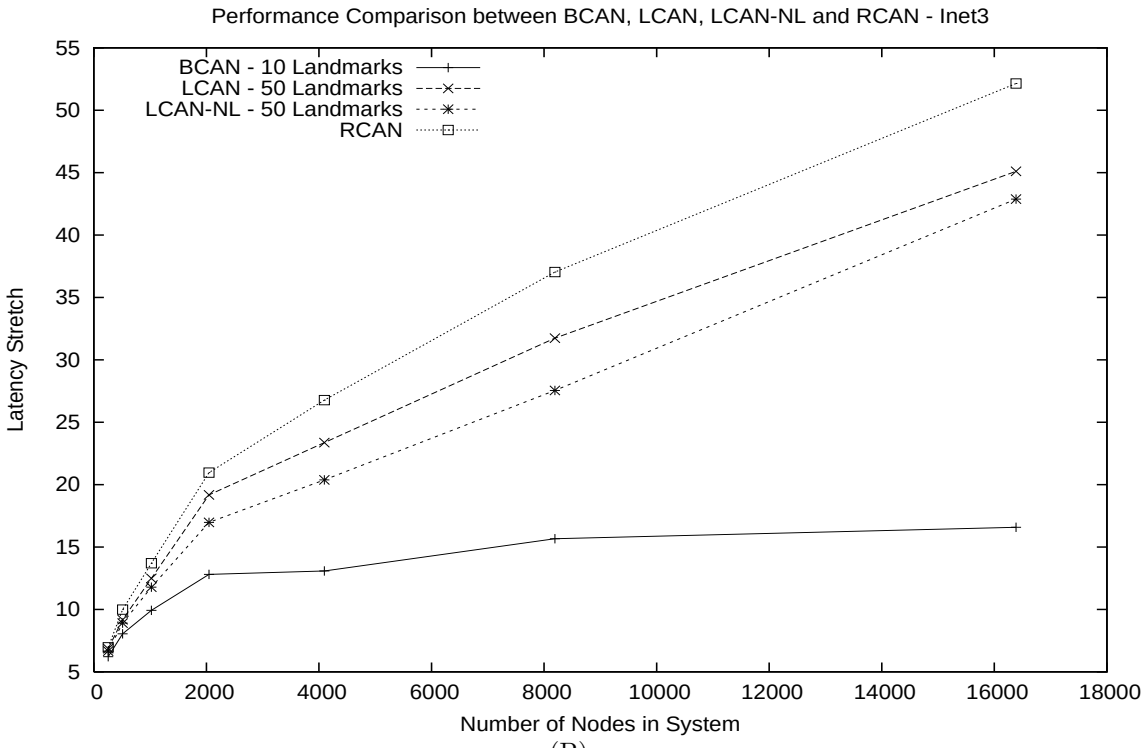
The latency stretch is obtained by finding the ratio of the end-to-end latency between randomly selected nodes in the overlay to the end-to-end latency on the underlying IP level network. We generate the system for 256, 512, 1K, ... 16K nodes. In case of BCAN we use 10 landmarks, and for LCAN we use 50 landmarks.

In LCAN, the load-balancing is achieved through stretching and resizing the zones according to the current population as explained in Section 4.2.2. While doing so, some nodes along the boundary of one zone may find themselves in another zone after resizing. Such nodes would not benefit from the binning, since they would be surrounded by nodes from a different zone. Hence there is a trade-off between load-balancing and performance here. We would like to see how much performance we are losing in order to achieve load-balancing. Hence we also present performance results for LCAN where the zone sizes are fixed and do not change for the duration of the simulation. We call this setup as LCAN-NL (No Load-balancing).

The results for LCAN, RCAN and BCAN are shown in Figure 5.2. BCAN outperforms LCAN in both topologies, which is expected due to the higher amount of selectivity in BCAN. But at the same time, LCAN also performs better than the RCAN, inspite of having the same load-balancing feature of RCAN (we shall see this when we compare the node degree distribution in a later section). The difference in performance for the two topologies can be explained by the distribution of latencies we saw in Figure 5.1. GT-ITM has a uniform distribution of latencies between 300-750 ms, whereas Inet3 has a high concentration around 200-350 ms and a short tail.



(A)



(B)

Figure 5.2: Latency stretch for BCAN, LCAN, LCAN-NL, and RCAN on GT-ITM (A) and Inet3 (B)

Intuitively, this means that in the Inet3 topology, most nodes are equidistant from each other with a small percentage of the remaining nodes set very far away. In such a situation, it is difficult to get a 2 dimensional embedding of the landmarks. This is why we see only a small improvement for Inet3 when using LCAN.

Another interesting observation is that there is very small improvement in LCAN-NL compared to LCAN. This suggests that the we do not lose much performance by maintaining the uniform load-balancing property of the RCAN.

5.4 Performance Variation with Number of Landmarks

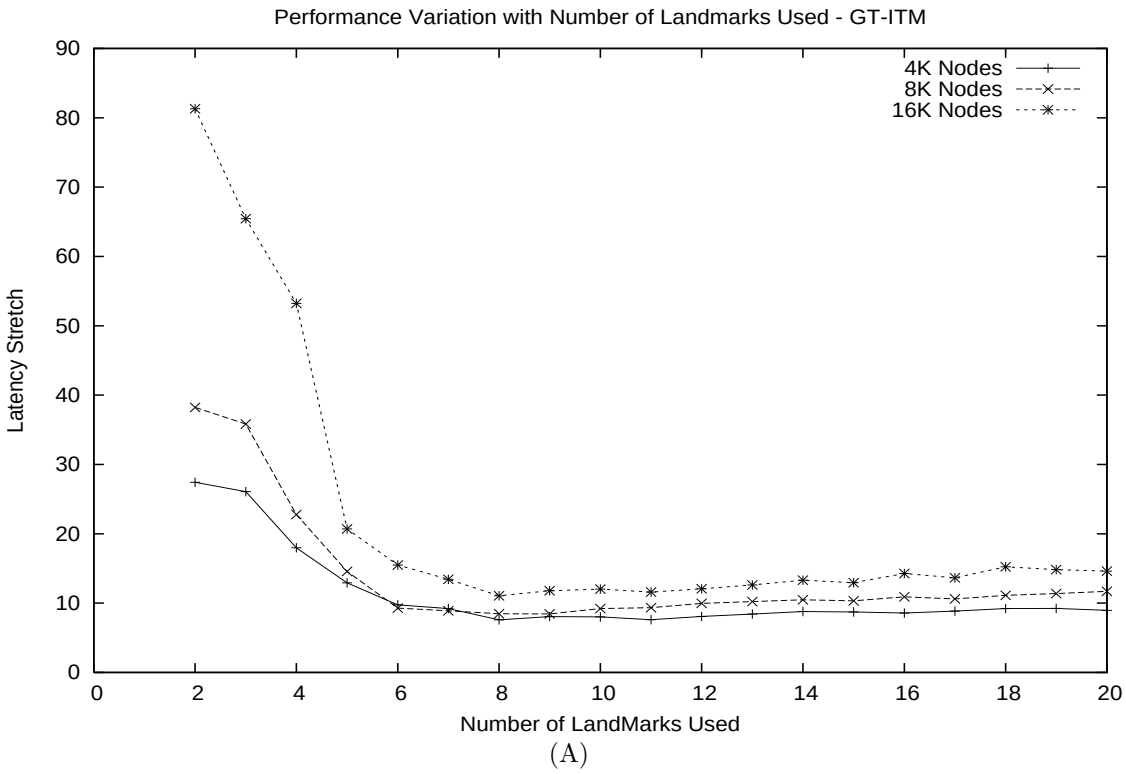
We create systems of 4K, 8K and 16K nodes using the BCAN and LCAN on both GT-ITM and Inet3 topologies. The number of landmarks is varied from 2 to 20. System performance is calculated in terms of the stretch observed between randomly selected nodes in the overlay. Figure 5.3 show that that in both topologies, the initial increase in the number of landmarks lowers stretch dramatically, but after achieving the optimal low stretch, further increase in number of landmarks actually degrades performance. There are two interesting things to note about from this experiment:

1. The optimal number of landmarks depends on the topology being used.

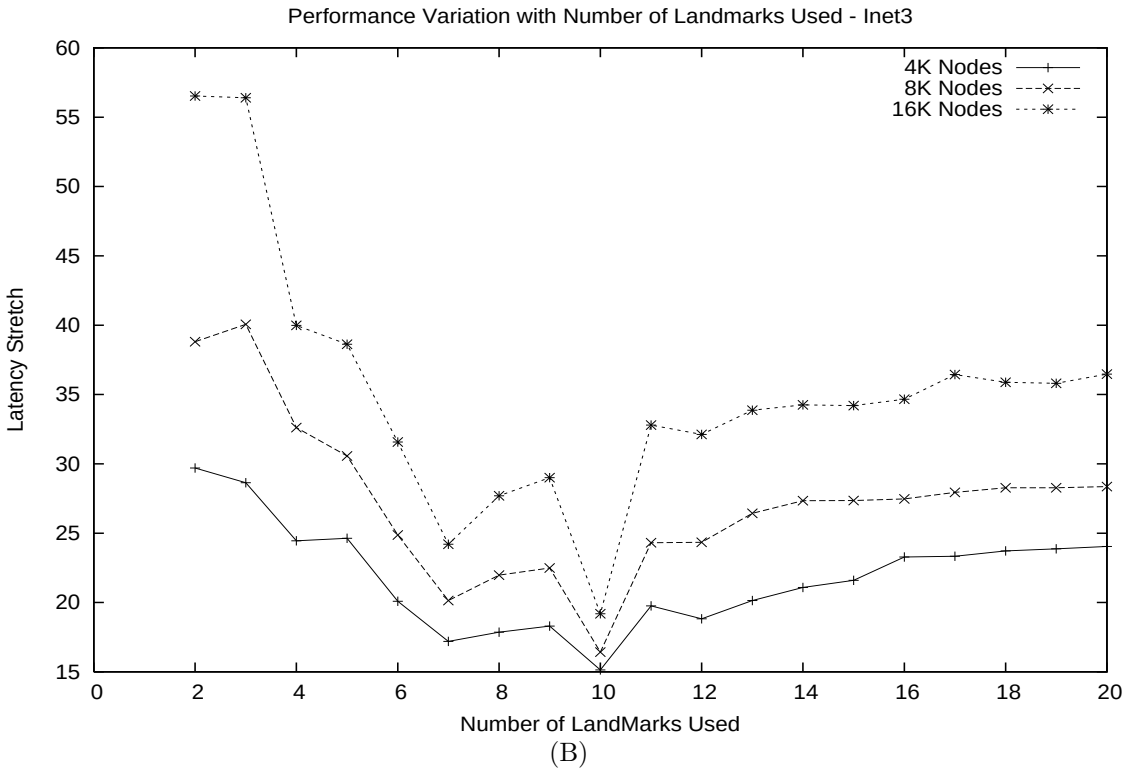
2. The degradation of performance with increase in landmarks is more pronounced in the Inet3 topology than in the GT-ITM topology.

With the Internet topology changing dynamically, it would be difficult to accurately predict the optimal number of landmarks. This is a significant disadvantage of the binning scheme.

We now look at the corresponding performance of LCAN with respect to variation in the number of landmarks. We again use systems with 4K, 8K and 16K nodes and

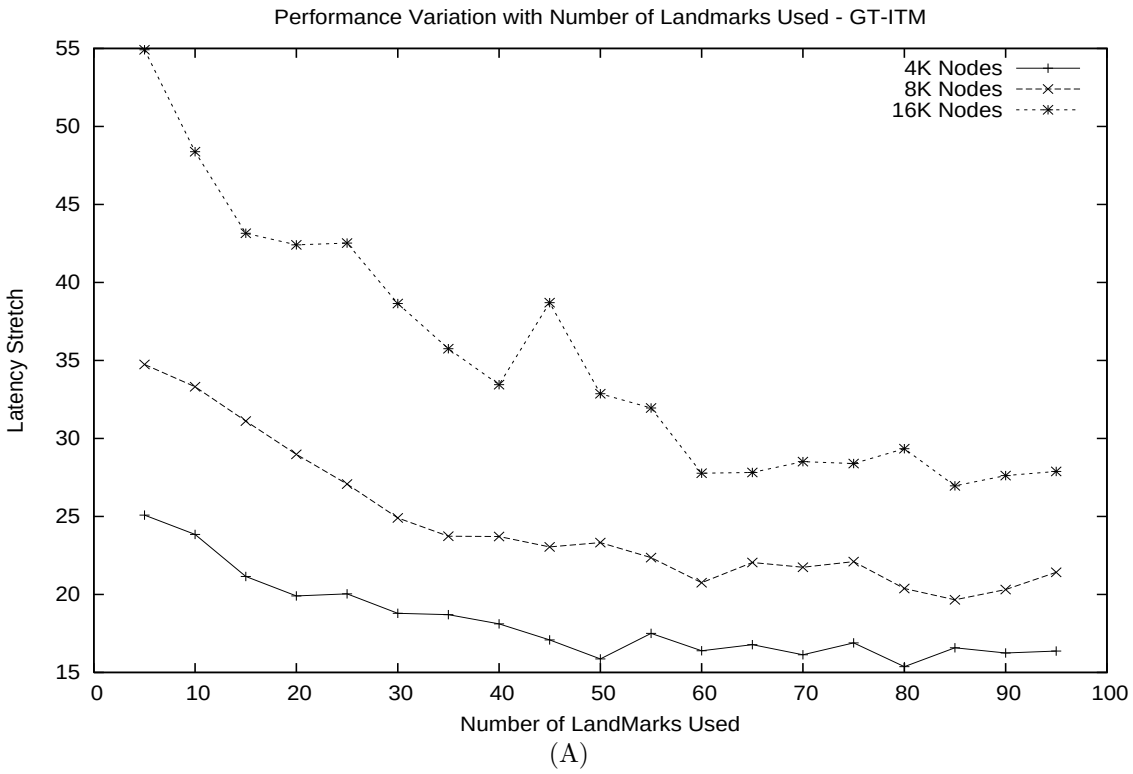


(A)

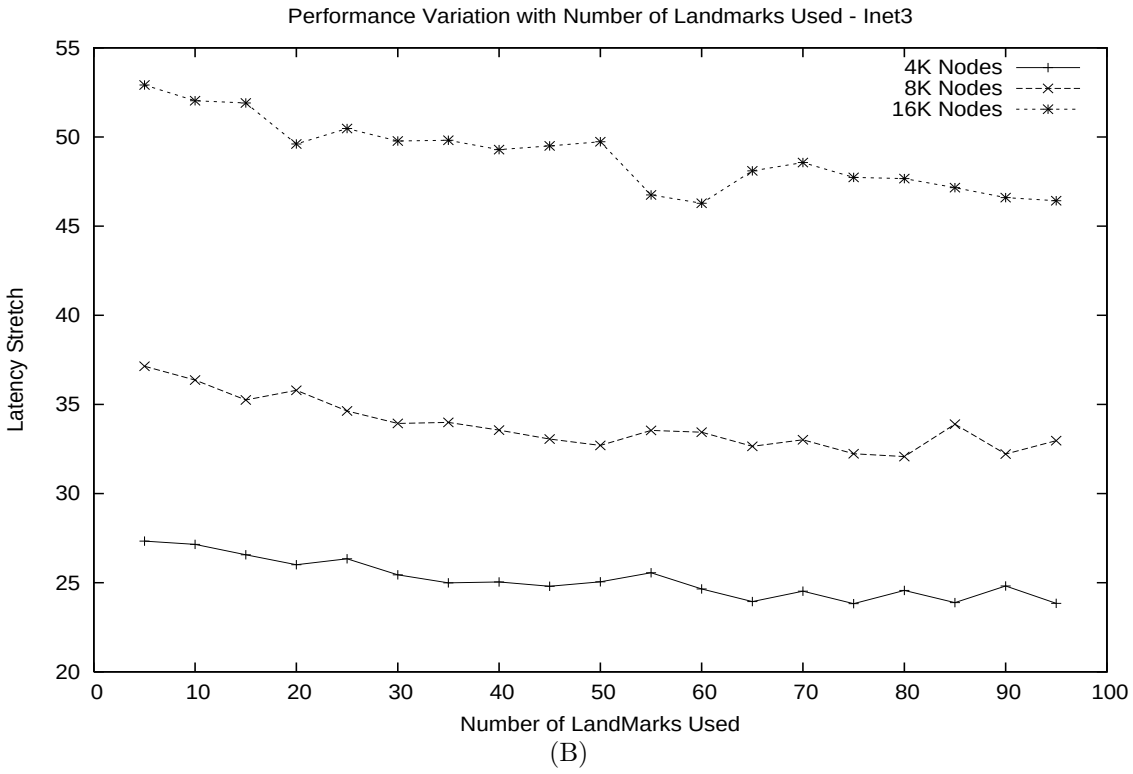


(B)

Figure 5.3: Variation of stretch with number of landmarks for BCAN on GT-ITM (A) and Inet3 (B)



(A)



(B)

Figure 5.4: Variation of latency stretch with number of landmarks for LCAN on GT-ITM (A) and Inet3 (B)

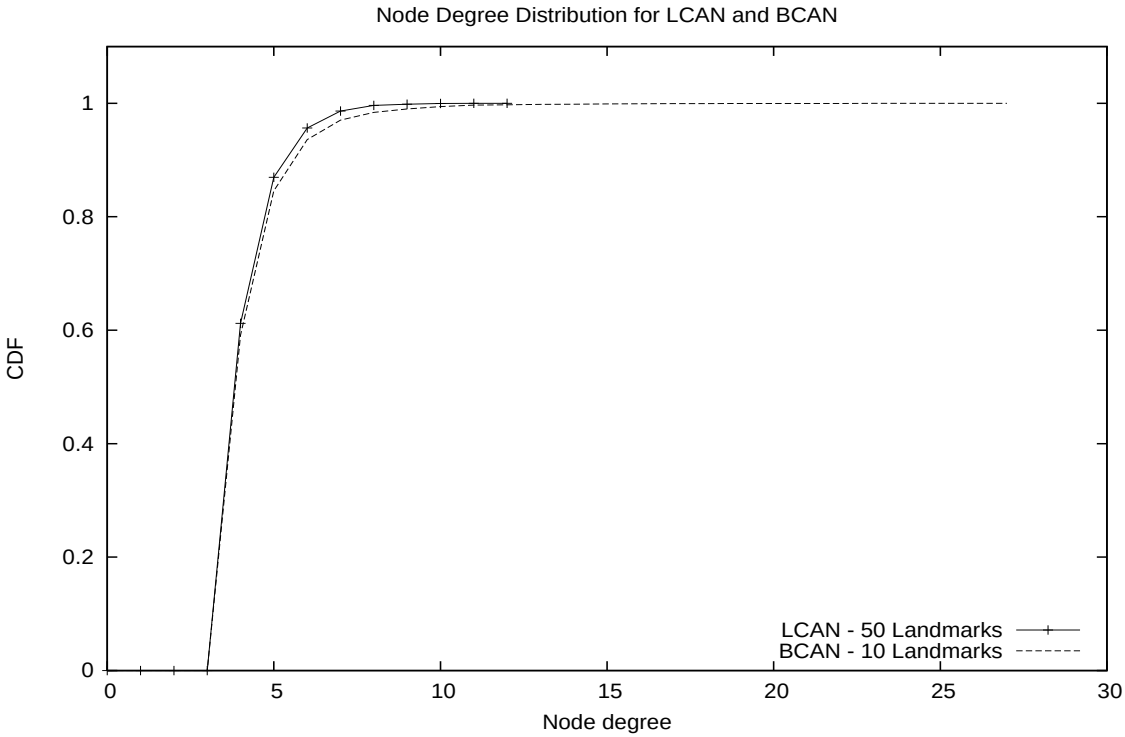


Figure 5.5: Node degree distribution for LCAN and BCAN

vary the number of landmarks from 5 to 100. From Figure 5.4 and , we find that the performance of LCAN does not degrade when we use more number of landmarks, in fact more is always better. This is a significant advantage of LCAN over BCAN. One important thing to note is that the improvement in performance for LCAN over Inet3 is not significant. This reinforces our observation in the previous section about the effect of topology on LCAN performance. The set of landmarks all being mostly equidistant from each other, we are not able to embed them onto the 2 dimensional CAN properly.

5.5 Node Degree Distribution

In this section we compare the state information maintained by each node in the form of neighbor table size, for LCAN and BCAN. As mentioned in Section 3.2, by

using BCAN, we lose the load balancing property of a DHT, and some nodes are burdened with more responsibility than others. On the other hand, as explained in Section 4.2.2, we make specific effort to preserve this load balancing aspect of the DHT in LCAN without affecting the performance improvement obtained by binning. We present below the results for the node neighbor table size (degree) distribution for LCAN (50 landmarks) and BCAN (6 landmarks) for a system with 10K nodes. The Figure 5.5 shows that we have been successful towards this goal. The LCAN distribution is what we would expect from an ordinary 2 dimensional RCAN. Although the BCAN shows similar distribution, some nodes do have degrees as high as 27.

Chapter 6

The Hypercube Binning Scheme

6.1 Introduction

In the previous chapter, we looked at a technique to make use of the delay to the closest landmark for labeling a node with an ID. This approach enabled us to provide better performance than RCAN. Now we explore whether we can do better by using second and third closest landmarks to improve the overlay organization. For using such higher dimensional information, we need an overlay with more dimensions. Hence in this section we introduce a PIL technique for the *Hypercube* DHT overlay.

A *k-cube* or a **Hypercube** of k dimensions is an undirected graph $H = (V_H, E_H)$ consisting of $n = 2^k$ vertices labeled from 0 to $n - 1$, such that there is an edge between any two vertices if and only if the binary representations of their labels differ by exactly one bit.

The motivation of using the Hypercube is twofold. Firstly, it provides a data structure with high connectivity. This makes it highly robust to failures. Secondly, the overlay distance between two nodes in the Hypercube is encoded in the *hamming distance* between their binary labels.

Let θ_j be the binary label of node j in the Hypercube and $\theta_j(i)$ be the bit at the i^{th} position in the binary label of node j. Then, the **hamming distance** between

two labels θ_a and θ_b of length k bits is defined to be $\text{HD}(\theta_a, \theta_b) = \mid \{i \mid \theta_a(i) \neq \theta_b(i), 1 \leq i \leq k\} \mid$.

If we could "map" the bins created by the Binning Scheme onto the nodes of the Hypercube, preserving this distance relationship, we would reap the benefits of not only the closeness of nodes within the same bin, but also the advantage of having the bins ordered on the overlay in close congruence with the physical topology. This mapping is achieved in two steps:

1. We *embed* the landmark graph onto the Hypercube.

   An **embedding** $\langle f, g \rangle$ of a graph G into a Hypercube H is defined by a mapping f from the nodes of G to the nodes of H, together with a mapping g that maps every edge $e = (v, w)$ of G into a path $g(e)$ connecting $f(v)$ and $f(w)$, the images of nodes v and w, in H.

   The binary labels assigned to the mapped landmarks on the Hypercube will have the property that two landmarks close together in the landmark graph will also be close together in the Hypercube.

2. Nodes joining the system will find their bin through delay measurements to the landmarks. In the CAN Binning Scheme, one may recall that corresponding to each bin is a particular ordering of the landmarks. In a similar manner, the binary label of each bin in the Hypercube is constructed by concatenating the binary labels assigned to each landmark, in the order corresponding to that bin.

The number of dimensions of the Hypercube is determined by the length of this label and the number of such landmarks appended together. For a *k-cube*, there may not be 2^k occupied bins at any stage during the life-time of the overlay. This is because of two reasons:

1. Certain bins may not be physically possible due to the constraints set by the underlying topology.

2. Nodes may not be present in the physical topology in locations which make some bins possible.

Hence some of the existing bins may need to *stretch out* and occupy a *volume* within the Hypercube enveloping their assigned label. Hence each bin in the *k-cube* is identified by two labels:

1. Point label, $\theta = \langle a_1 a_2 ... a_k \rangle$, $a_i \in \{0, 1\}$, $1 \leq i \leq k$
2. Volume label, $\phi = \langle a_1 a_2 ... a_k \rangle$, $a_i \in \{0, 1, X\}$, $1 \leq i \leq k$

The Point label denotes the location of the bin within the Hypercube, whereas the Volume label denotes the volume occupied by that bin. The X or "don't care" terms in the Volume label denote the unsplit dimensions of that bin. A single X term implies that the bin *stretches* across an edge of the Hypercube, two X terms mean that the bin *stretches* across a $4-$ cycle within the Hypercube, and so on. Hence a bin b in the Hypercube is represented by $\{b, \phi_b, \theta_b\}$, where ϕ_b is its volume label and θ_b is its point label. If the identity of a bin is clear from the context, we may skip it and represent the bin merely as $\{\phi_b, \theta_b\}$.

The introduction of don't care terms in the volume label necessitates the redefinition of the hamming distance between two nodes as follows:

The **hamming distance** between two volume labels ϕ_a and ϕ_b of length k bits is defined to be $\text{HD}(\phi_a, \phi_b) = \mid \{i \mid \phi_a(i) \neq \phi_b(i),\ \phi_a(i), \phi_b(i) \neq X,\ 1 \leq i \leq k\} \mid$.

Two properties of the Volume label and the Point label will be of use to us further on in this section:

Containment Property: $\text{HD}(\theta, \phi) = 0$: This property states that point label of the Hypercube lies completely within volume label, and hence the hamming distance between them must be 0.

Completeness Property: If the number of bins in the k cube is 2^k, then $\theta = \phi$ - This property states that for the case when all the bins within the Hypercube have

been completely occupied, no bin needs to *stretch* across any dimension, hence there are no X terms in the Volume label, and the Point label and the Volume label are identical.

6.2 Assignment of Binary Labels to the Landmarks

Binary labels must be assigned to landmarks in such a manner that the hamming distance between bins should reflect the physical distance between them. To achieve this, we first embed the graph of landmarks into the Hypercube.

The quality of an embedding is evaluated by two properties: *dilation* and *expansion*, defined as below:

The **dilation** δ is defined as the maximum distance in H between images (in H) of two adjacent nodes in G when mapped under g.

The **expansion** ε is defined to be the ratio of the number of nodes in H to the number of nodes in G.

Intuitively, the dilation is a measure of how much discrepancy there is between the graph and its embedding, in terms of distance while the expansion is a measure of how many extra nodes are required to achieve the embedding. We are specifically interested in embeddings with dilation one. When a landmark graph is embedded into the Hypercube with dilation one, landmarks which are adjacent in the landmark graph, and hence close together in the physical topology, will also be adjacent in the Hypercube. Also, as discussed before, the length of the landmark label decides the number of dimensions of the Hypercube, which in turn decides the amount of information each node will have to maintain. Hence our motivation in this section would be to achieve an *optimal embedding*.

An **optimal embedding** of a graph would be one with expansion one and dilation less than two.

As an example, consider a graph of 16 landmarks. The optimal embedding would embed this graph into the 4-cube. But an embedding with expansion two, would be an embedding into a 5-cube. This requires one bit more than the optimal embedding. Thus an increment of one in the expansion translates to one extra bit required for the labeling.

Unfortunately, for arbitrary graph G with unbounded degree, it is NP-Complete to decide whether it can be optimally embedded into a k-cube [23]. The unbounded degree of the graph is essential for proving NP-Completeness. If the graph G has bounded degree, efficient embeddings are possible. The best results for bounded degree graphs in terms of dilation and expansion are for the binary trees. Of particular interest to us is a conjecture by Havel [10], which states that all *parity-balanced binary trees* with 2^k vertices can be embedded optimally in a k-cube.

A **parity-balanced tree** is a tree in which the two bi partitions of the tree have the same number of vertices.

We make use of results in [4], in which it is also proved that any binary tree may be parity-balanced by adding extra leaves to the smaller bi-partition. Thus a binary tree T with n vertices may be balanced by adding no more than n vertices. Together with Havel's conjecture this result gives us an embedding for arbitrary binary trees with dilation 1 and expansion no more than 2.

Given such a parity-balanced binary tree, we can run any simple greedy search algorithm to get the embedding of this tree into the Hypercube. The leaves added to make the tree balanced are discarded later on. For all simulation results in this thesis, we have used the algorithm given in [1].

Hence, given this information, we proceed to assign binary labels to the landmarks in the following manner:

1. From the landmark graph, obtain the binary spanning tree. This is obtained by following the usual Kruskal algorithm for spanning trees, but if the addition of an edge results in the degree of any node increasing beyond 3, that edge is

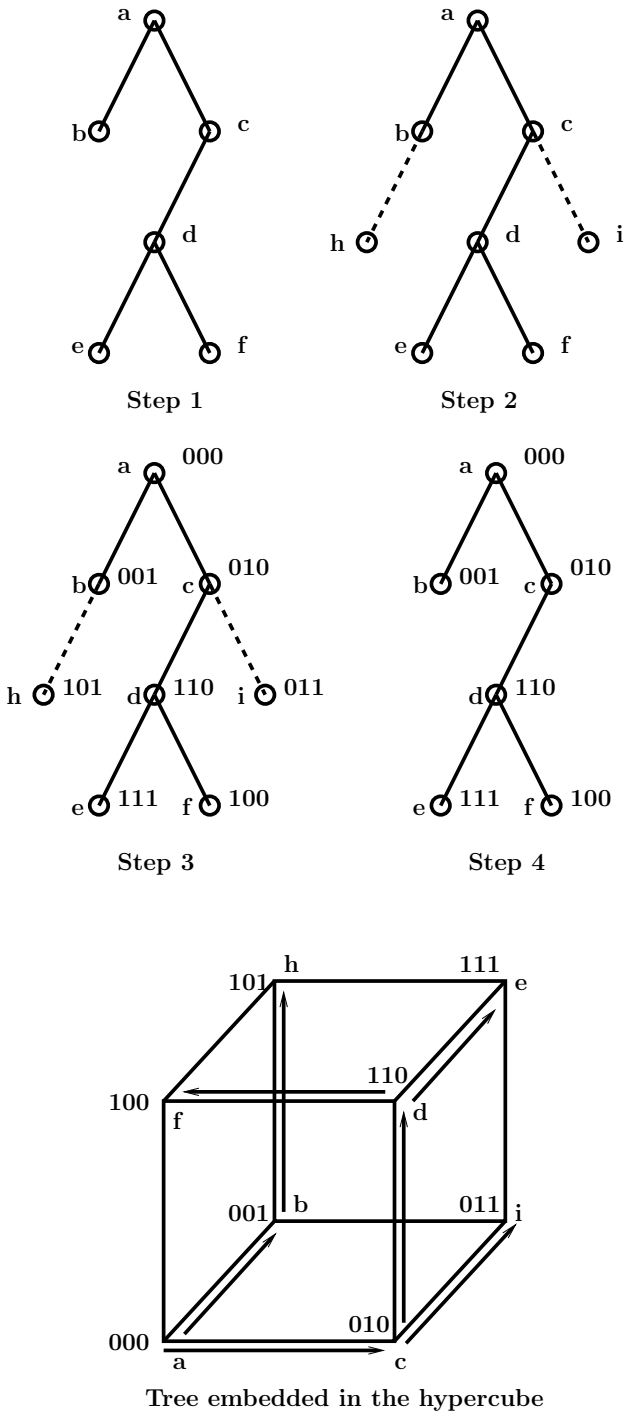


Figure 6.1: Example of embedding the landmark graph on the Hypercube

discarded and the algorithm continues with the next shortest edge.

2. This binary spanning tree is converted to a balanced binary tree, by adding the required number of leaves to the smaller bi-partition.

3. The optimal embedding is obtained for this tree, using the greedy search.

4. The leaves added in step 2 are discarded, and the labels of the images in the Hypercube become the required binary labels of the corresponding landmarks.

These steps are illustrated in the example in Figure 6.1. We use a system of 6 landmarks, a, b, c, d, e and f for this example. Using the modified Kruskal's algorithm, we generate a binary spanning tree which is shown in Step 1. Now we observe that this tree is unbalanced. The two bi-partitions are composed of $\{a, d\}$ and $\{b, c, e, f\}$. To make it balanced, we add two dummy nodes, h and i to b and c respectively. In step 3, we run the greedy search algorithm to get the optimal labeling for this balanced binary spanning tree, and the resultant labels are as shown in the figure. Finally in step 4, we discard the dummy nodes. Note that the dummy nodes are always leaves and their adding/discarding has no effect on the dilation of the mapping. The spanning tree as seen in the 3-cube is also shown in the figure. One important property of this assignment is that since the dilation is 1, the adjacency property is satisfied, i.e. if two landmarks are close to each other, and are adjacent in the binary spanning tree, they would also be adjacent in the Hypercube. But two nodes being adjacent in the Hypercube would not necessarily imply that they are also close in the underlying topology. Consider the nodes f and h in the Hypercube. They are very far away in the spanning tree, but in the Hypercube, they are adjacent to each other. This property is used of during the routing in the Hypercube of bins.

6.3 Assignment of Point Labels to Bins

Each newly joining node finds its bin by measuring delays to the set of landmarks. The point label of the bin of this node is formed by concatenating together the landmark labels in the increasing order of the delays. The number of landmark labels appended is a configurable parameter λ. Consider the previous example - a node which is newly joining the Hypercube finds the delays to the landmarks a, b, c, d, e and f and finds that the delay order is: c, a, b, f, e, d. Then the corresponding point label of the bin of this node would be formed by appending together the first λ landmark labels. With a $\lambda = 3$, we get a point label of 010000001. Hence the resulting Hypercube of the bins would be a 9-cube.

There is a trade-off between efficiency and state information when we change the value of λ. With higher values of λ, we are incorporating more delay information, and hence we should be able to create better overlays, but at the same time, the bin Hypercube dimensions would be larger and hence would lead to each bin having to maintain more number of neighbors.

Another problem is that due to nodes joining in a non-uniform manner some bins may have higher population than other bins. The nodes mapped to the same bin will all have the same volume and point labels and would require the election of a bin leader or some other cluster management technique. We avoid this by appending a random binary string to the point and volume labels with enough bits (μ) to accommodate all such nodes binned to the same location in the Hypercube. The consequence of this is that the dimensions of the Hypercube will be further increased and each node may have to maintain a small number of extra neighbors corresponding to the random bits. Again we encounter a trade-off between load-balancing and efficiency. By increasing the size of the random bit string we can balance the load among the nodes, whereas the locality of the Hypercube suffers. Hence we need to achieve a balance between the locality (λ), and load-balancing (μ). The addition of the random bits ensures

that every node in the Hypercube has unique point and volume labels. Henceforth, we use the term bins and nodes interchangeably.

6.4 Routing in the Hypercube

Data : Message is to be routed from a source bin, $src = \{\theta_{src}, \phi_{src}\}$, to the bin responsible for θ_{dest}

Result: Message is routed to the bin responsible for the destination point in the Hypercube

$curr = src$;

while $HD(\theta_{dest}, \phi_{curr}) \neq 0$ **do**

/* Get prospective next hop neighbors for curr */

$nHopSet =$ { Neighbors of $curr$ with minimum hamming distance to θ_{dest}};

/* Select closest one among them */

$nextHop =$ Bin in $nHopSet$ closest to $curr$ in terms of delay;

Forward message to $nextHop$;

$curr = nextHop$;

end

return $curr$;

Algorithm 3: Hypercube Routing Algorithm

The Hypercube as an inter-connection network for parallel computers has been studied extensively. It has also been recently suggested as an overlay structure in Hypercast [12]. Routing in such Hypercubes proceeds as follows:

1. On receiving a message from X to be sent to Y, the node Z will get the hamming distance of all its neighbors in the Hypercube to the destination Y.

2. The node will get a set of neighbor nodes, all of which are at the same ham-

ming distance to the destination node Y. It can forward the message to any one of these prospective neighbors, because all of these paths will lead to the destination.

This routing algorithm, though very simple to implement, does not take into account the actual underlying topology and the position of the nodes in this topology. Our routing algorithm takes advantage of the fact that the bins in the Hypercube are actually ordered on the basis of their proximity to each other. Routing proceeds in two steps, the first of which is the same as in the original Hypercube routing, i.e. we find the set of neighbors which are at the minimum hamming distance from the destination.

Now, remember that according to the embedding in the previous section, a node in our Hypercube will have some neighbors which are not physically close because of the artifacts of embedding. So the next step comprises of weeding out such non-close neighbors and selecting the closest neighbor among the set of possible next hop neighbors. This is done on the basis of delay measurements. Finally, the message is forwarded to the closest neighbor. This algorithm is presented in Algorithm 3. The algorithm terminates when the hamming distance between the current node's volume label and the destination's point label is zero. This is due to the containment property, defined earlier. When this happens, we know that the message has arrived at the destination, since the hamming distance between a node's point and volume labels is always zero.

6.5 Join Procedure in the Hypercube

The join algorithm proceeds in the following manner. New nodes wishing to join the overlay will first find its point label in the Hypercube. It gets the address of a random node within the overly through a bootstrap server. It sends a message starting at this random node with the destination as its point label through Hypercube

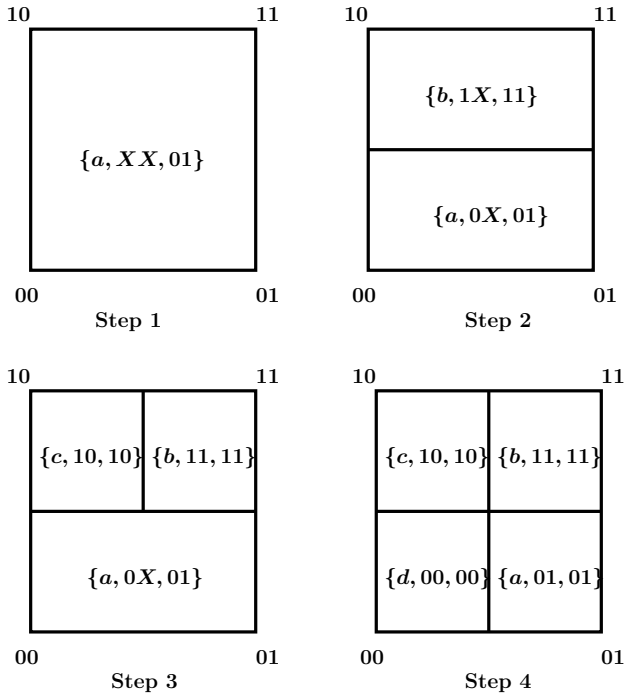


Figure 6.2: Example of nodes joining the 2-cube

routing.

On reaching the destination, if it finds a bin with the same point label, it joins that bin and the algorithm terminates. Else it will split a random unsplit dimension (indicated by don't care symbols in the volume label) at which their point labels differ. The update of neighbors has to be done carefully. Consider a bin which has been split along dimension i. Among its current neighbors, there are three possibilities:

1. Bins which are unsplit along dimension i. Such bins would have an X at position i in their volume id. Such bins are at a distance of 1 from the old bin as well as the newly joined bin, hence they remain neighbors of both bins.

2. Bins which differ in their volume id along dimension i. Such bins are no longer neighbors of the old bin, but would be a neighbor of the new bin, since the bin dimensions can only have one of two possible values, 0 or 1.

3. Bins which do not differ in their volume id along dimension i. Such bins are still neighbors of the old bin, but would not be a neighbor of the new bin, since they would be at a hamming distance of 2 from the new bin.

The join procedure is illustrated with an example in Figure 6.2. The 2-cube has 2 dimensions, hence it is capable of containing four bins, $\{00, 01, 10, 11\}$. Now, newly joining node $\{a, 01, XX\}$ will occupy the whole Hypercube, since it is the first node. This is represented by its Volume label, $\langle XX \rangle$, which tells us that both the dimensions of the 2-cube have yet to be split. This situation is shown in Step 1 in Figure 6.2.

Now, another new node b, with a hash id $\langle 11 \rangle$ joins the overlay. A join request for this hash id is inserted into the overlay, and since the node responsible for the whole hash space is a, it is chosen to be split. According to the Algorithm 4, the split occurs as follows. We first find the set of unsplit dimensions of a along which the hash ids of a and b differ. In this case, they differ only along the first dimension, and hence it is selected to be split. Then, any neighbors of a, (in this case none) are updated for this new situation.

The join request of nodes c and d proceed in a similar way.

The significance of the "sibling" nodes in Algorithm 4 will be clear in the next section when we discuss the leave procedure in the Hypercube.

6.6 Leave Procedure in the Hypercube

A node in the P2P system may either gracefully leave the system, after informing its neighbors or may die unexpectedly. The case where a node leaves gracefully, is discussed in section 6.6.1. The case where a node leaves unexpectedly, without informing its neighbors is discussed in section 6.6.2.

Data : Newly joining node with bin $\{a, \theta_a, \phi_a\}$, ϕ_a is unknown
Result: Node inserted into k-cube
s = random node address from bootstrap server;

/* Get the bin responsible for the point θ_a */
$\{b, \theta_b, \phi_b\}$ = HypercubeRoute(s, θ_a);

if $\theta_b == \theta_a$ **then**

    Add node to this bin and return;

end

/* Get the set of dimensions which can be split */
$splitSet = \cup_{i=1}^{i=k}\{\ i \mid \theta_b(i) \neq \theta_a(i) \cap \phi_b(i) = X\}$;

Let j = A randomly selected dimension from the set $splitSet$;

/* Volume id of a is same as that of b */
$\phi_a = \phi_b$;

/* Except for the the bit where the split occurs */
$\phi_a(j) = \theta_a(j)$, $\phi_b(j) = \theta_b(j)$;

foreach *neighbor bin* $\{n, \theta_n, \phi_n\}$ *of* b *in Hypercube* **do**

    if $\phi_n(j) == X$ **then**

        /* n is the neighbor of both a and b */
        Add n as a neighbor of a and vice versa;

        If b is in the sibling set of n, add a into it;

    else if $\phi_b(j) \neq \phi_n(j)$ **then**

        /* this bin is no longer neighbor of b */
        Remove n as a neighbor of b and vice versa;

        Add n as a neighbor of a and vice versa;

end

Add a and b as siblings and neighbors of each other.

Algorithm 4: Node Join Algorithm.

Data : Node a with bin $\{\theta_a, \phi_a\}$, and sibling set S wants to leave the system
Result: Node deleted consistently from k-cube
/* `Send message to to all nodes in` S `to expand and fill up volume` θ_a. */
foreach *neighbor* $\{n, \theta_n, \phi_n\}$ *of* a *in* S **do**
    /* `Find differing bit and expand to` X */
    Find $i \mid \phi_n(i) \neq \phi_a(i)$;
    $\theta_n(i) = X$;
end
Connect neighbors of a who, after expansion will be neighbors of each other.
Leave the system.

Algorithm 5: Graceful Leave Algorithm.

6.6.1 Graceful Leave

To understand how to achieve a correct leave, let us review the node join procedure once again. When a node joins the overlay, it does so by splitting the space belonging to one already existing node. The old node splits one of its X terms to either 1 or 0.

Now, these two nodes are eligible to expand into each other's space if either of them fails, by just "unsplitting" the same bit back to X. We will call such nodes as "sibling" nodes. Hence each node just needs to know the last node that it split, so that if it wants to leave, it will contact that node to occupy its space again. There is one subtle aspect. If any node a finds that its sibling b has been split and the new node, c, which caused the split is also its neighbor (recall bins of type 1 in the previous section), then now a considers both b and c to be its siblings. We will call the set of such siblings to be the "sibling set". When a node wants to leave, it needs to inform all the nodes in the sibling set to expand and fill up the space left by it. The sibling set is reset every time a node gets split again, and is replaced with the new sibling. This procedure is presented in Algorithm 5.

We illustrate the algorithm with an example. Going back to Figure 6.2, in Step 1, node a's sibling set is empty. In step 2, node b splits node a's space in the 2-cube, hence a and b are now siblings of each other. In Step 3, c splits b. This causes b to reset the current sibling set containing a and replace it with the new sibling set containing c. b is also put into the sibling set of c. The case of a is more interesting. Since the node which has just been split is b, which is in a's sibling set, and since both b and c are still neighbors, a will add c to its sibling set. Finally, in step 4, d splits a's space, and hence a resets its sibling set containing b, c and replaces it with the sibling set containing just d. Hence we are left with the situation where a, d and b, c are siblings of each other. With this information, graceful leave of any single node is handled by its corresponding sibling node. We note that we maintain only enough information to handle single leaves. To consistently handle multiple consecutive leaves, one needs to maintain extra state information. Consider what happens once node a wants to leave the system. d will expand into a's space. But now d does not have a sibling node. Now if it wants to leave the system it must find its sibling node among its current neighbors. This is done by selecting the set of neighbors which most closely match its volume label in terms of X terms.

6.6.2 Abrupt Leave

We now look into the case of the abrupt leave. Being a distributed system, a node's knowledge about the system is limited to just one hop neighbors. The neighbors of the failed node will each try to occupy some space of the failed node, and propagate this information to their own neighbors. If two neighbors of the dead node decide to occupy some common space belonging to the failed node simultaneously, the Hypercube will be inconsistent. We employ a reactive approach where each node tries to fill up as much of the failed node's space as possible, and backs off if it detects a collision. [17] employs a similar try/back-off approach for its node leave procedure.

One benefit of the Hypercube is that because of its high connectivity, it is highly

robust to node failures. It would take a large number of simultaneous failures for the Hypercube topology to get disconnected.

Chapter 7

Hypercube Simulation Results

In this section we look at the performance of the Hypercube structure in terms of stretch and node state information as we vary the number of landmarks. We compare results with RCAN. Among the existing P2P systems, CAN [17] has the property that with increase in the number of dimensions, we see an improvement of the overlay stretch at the cost of increased state information. We run a simple experiment to find the best possible performance available from the CAN P2P system with increasing dimensions, and the results are in Figure 7.1. We use a range of dimensions from 5-50 in a system with 10K nodes. We see that after a significant improvement from 5 to 20 dimensions, the performance is more or less same regardless of increase in the number of dimensions. As seen in the figure, this is true for both GT-ITM and Inet3 topologies. Hence we use 20 dimensions for all results involving RCAN.

Please note that this is an RCAN with no knowledge of the underlying topology. We have used RCAN for comparison purposes instead of BCAN because of a practical issue. We have demonstrated in Section 5.4 that BCAN performance degrades after 10 landmarks. The embedding scheme in [18] required distributing landmarks in a cyclic fashion, to distribute the information evenly among all the dimensions equally. Hence there is an implicit assumption that the number of landmarks is more than the number of dimensions of the CAN. With the number of landmarks limited to 10, we

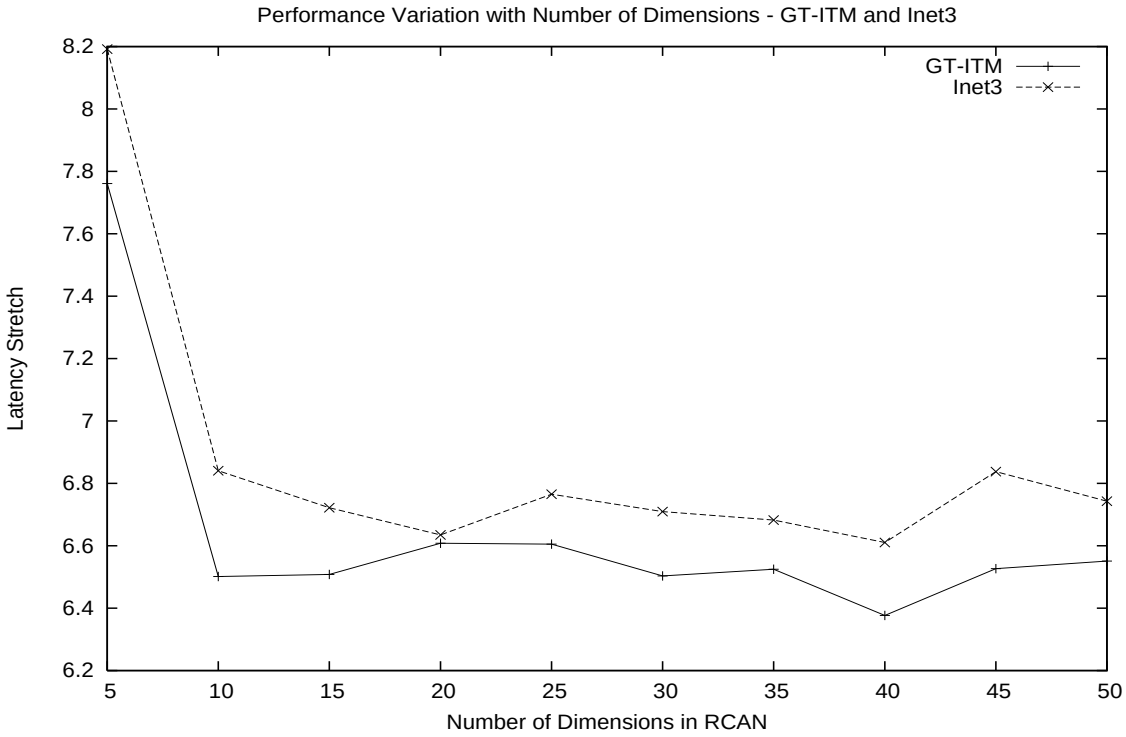


Figure 7.1: Variation of latency stretch with dimensions in RCAN

cannot use BCAN with very high dimensions to match the required amount of state information held by the Hypercube.

We also compare the performance with a random Hypercube (RCUBE), with the same number of dimensions as the Hypercube with landmarks (which we will henceforth call the LCUBE), to understand how much we gain by the use of landmark embedding.

The Hypercube has three parameters which decide its performance:

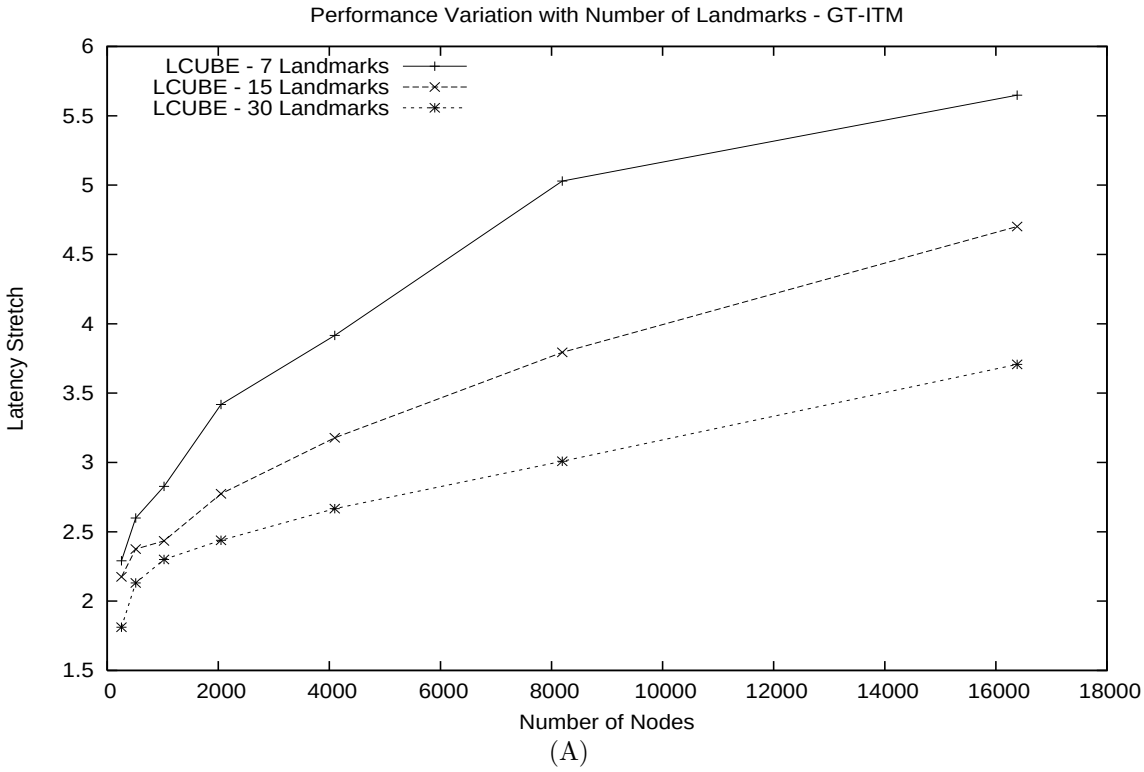
1. **The Number of Landmarks (n)** With an increase in the number of landmarks, we get a more comprehensive view of the network. Our embedding technique guarantees that for a system of n landmarks, we obtain an embedding of $\lceil log(n) \rceil$. Note that the dummy nodes added for the sake of labeling, may increase the total length of the labeling by at most one bit. This indicates

that for complete use of all binary labels, n should be as close as possible to a power of 2. We may have to run the labeling algorithm multiple times with different selection of landmarks to achieve this, but being an offline calculation, this is not an issue.

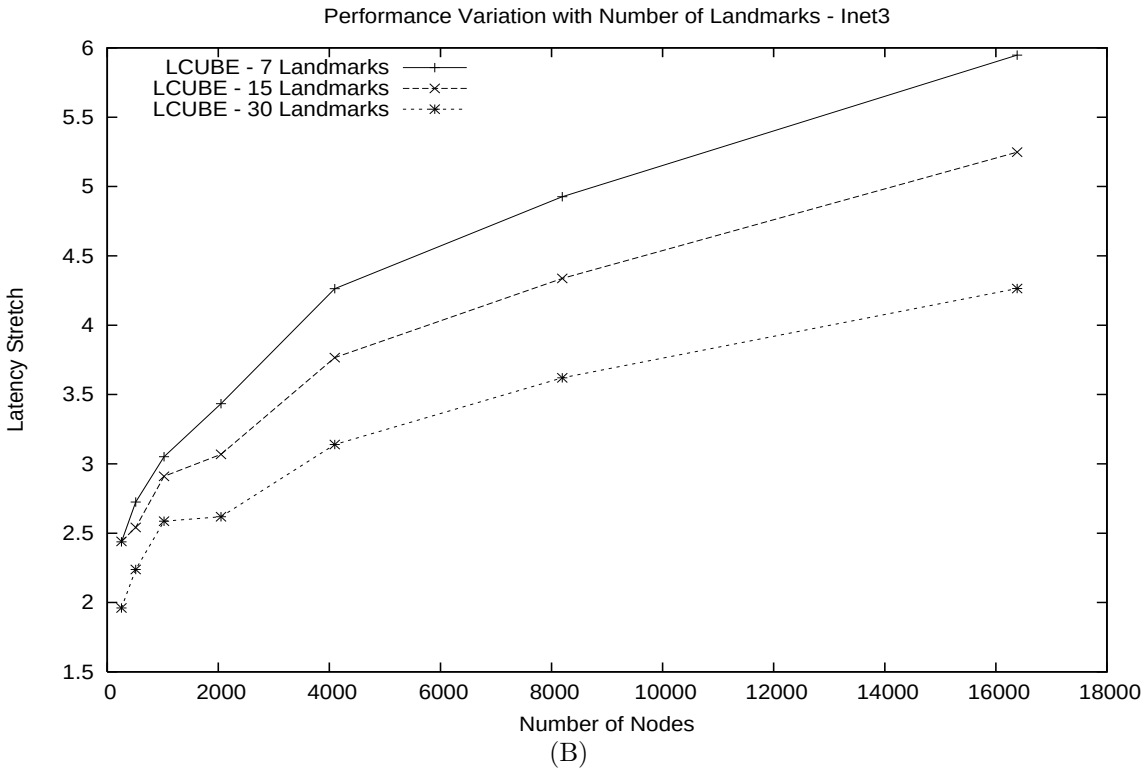
2. **The Number of Landmark Labels Appended for Hash ID (λ)** As discussed earlier, with increasing λ we see an increase in the locality information available to the system, and hence a corresponding increase in the performance of the system in terms of end-to-end latency. But with this we also see an increase in the number of dimensions of the Hypercube leading to an increase in the state information each node has to maintain, e.g. neighbor table sizes.

3. **The Number of Random Bits Appended (μ)** We append μ random bits behind the landmark labels. This will randomize the nodes binned into the same location in the Hypercube. Hence such nodes will have the same prefix, formed by the λ landmark labels together but they will be differentiated by their random bits. It is the random bits which allow the Hypercube to scale without increasing state information. If a node joins the overlay with a particular label, and finds that a node already exists with that label, it could rejoin with the same landmark prefix but a new random suffix.

From the above parameters, we find that a system with n landmarks, λ landmark labels appended, followed by μ random bits would result in a Hypercube with $(\lceil log(n) \rceil * \lambda) + \mu$ dimensions. Our goal would be to keep the dimensions as low as possible and maintain good performance.



(A)



(B)

Figure 7.2: Variation of latency stretch with number of landmarks for LCUBE on GT-ITM (A) and Inet3 (B)

7.1 Performance Variation with the Number of Landmarks (n)

We use a system with number of nodes ranging from 256, 512, 1K, ... 16K, and observe the results for $7, 15$ and 30 landmarks, with $\lambda = 1$, $\mu = 10$. The Figure 7.2 shows the stretch for each of the cases. We see that as expected, the increase in number of landmarks does bring about corresponding decrease in the stretch. Recall that the Laminar Binning technique did not provide much improvement with more landmarks when used on the Inet3 topology, since the algorithm was dependent on the latency distribution of the underlying topology. In Figure 7.2 (B), we see that this is not the case with the LCUBE. The performance of LCUBE gets better with more landmarks, while it is true that the performance improvement is slightly more in the GT-ITM topology.

7.2 Performance Variation with the Number of Landmarks Appended (λ)

The number of landmarks appended decides the amount of locality information available to the system. We run the simulation with $n = 30$ landmarks using $\lambda = 1, 2$ and $\mu = 10$ and create a system with 256, 512, 1K, ... 16K nodes. The corresponding Figure 7.3 shows the large performance gap between the two cases. Note that with just 2 appended landmarks, we are able to achieve a stretch of 4 for a 16K node system. Of course, this comes at the cost of node state information. For the case when two landmark labels are appended, some nodes have to maintain as high as 200 nodes in their neighbor table. This large overhead makes appending more than one landmark only feasible for large overlays comprising of highly connected nodes. For small overlays not overly concerned about latency, the performance from a single

landmark label would suffice.

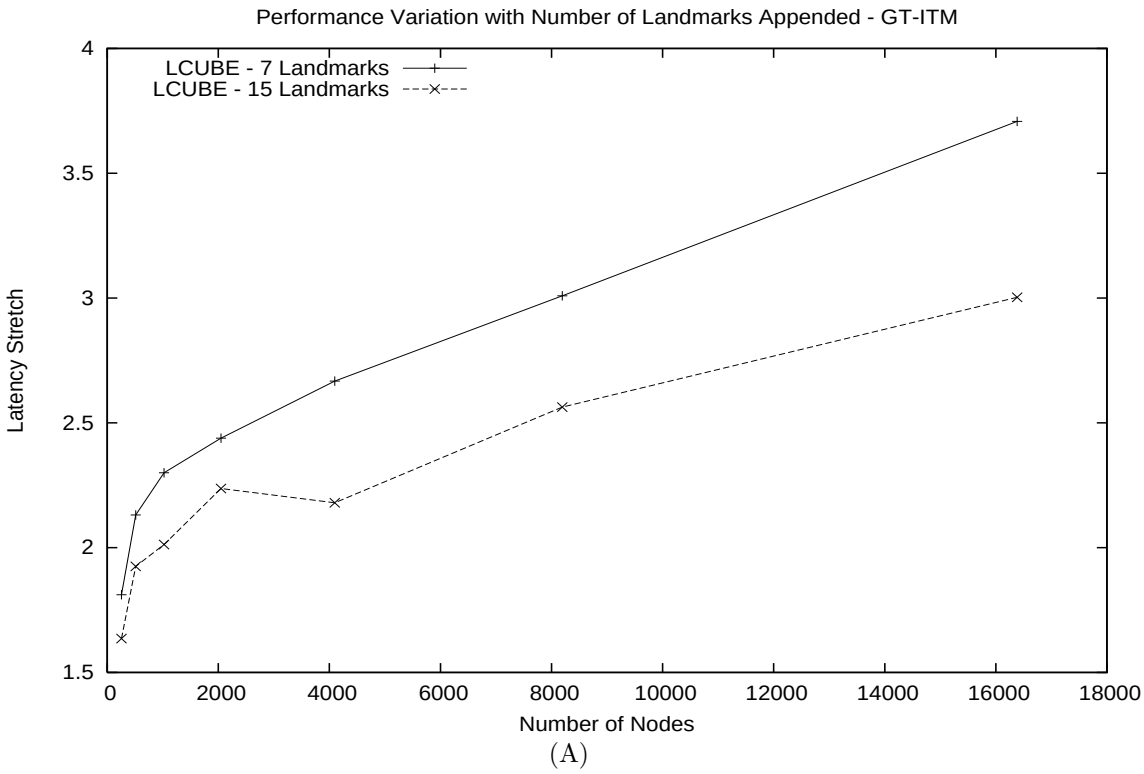
7.3 Latency Stretch

Here we compare the performance of the Hypercube with the RCAN with 20 dimensions, an LCUBE with $n = 30$, $\lambda = 1$ and $\mu = 10$, and the RCUBE. This gives us the idea of how the Hypercube performs compared to a RCAN with about the same amount of state information. For RCUBE, we first run the embedding algorithm for $n = 30$, $\lambda = 1$ and $mu = 15$, and then use the resultant number of dimensions for its construction. This allows a fair comparison between RCUBE and LCUBE, since each node maintains the same amount of state in either overlay. The Figure 7.4 shows the results of the experiment. As can be seen, the RCUBE itself far outperforms the RCAN and LCUBE also outperforms the RCUBE in turn.

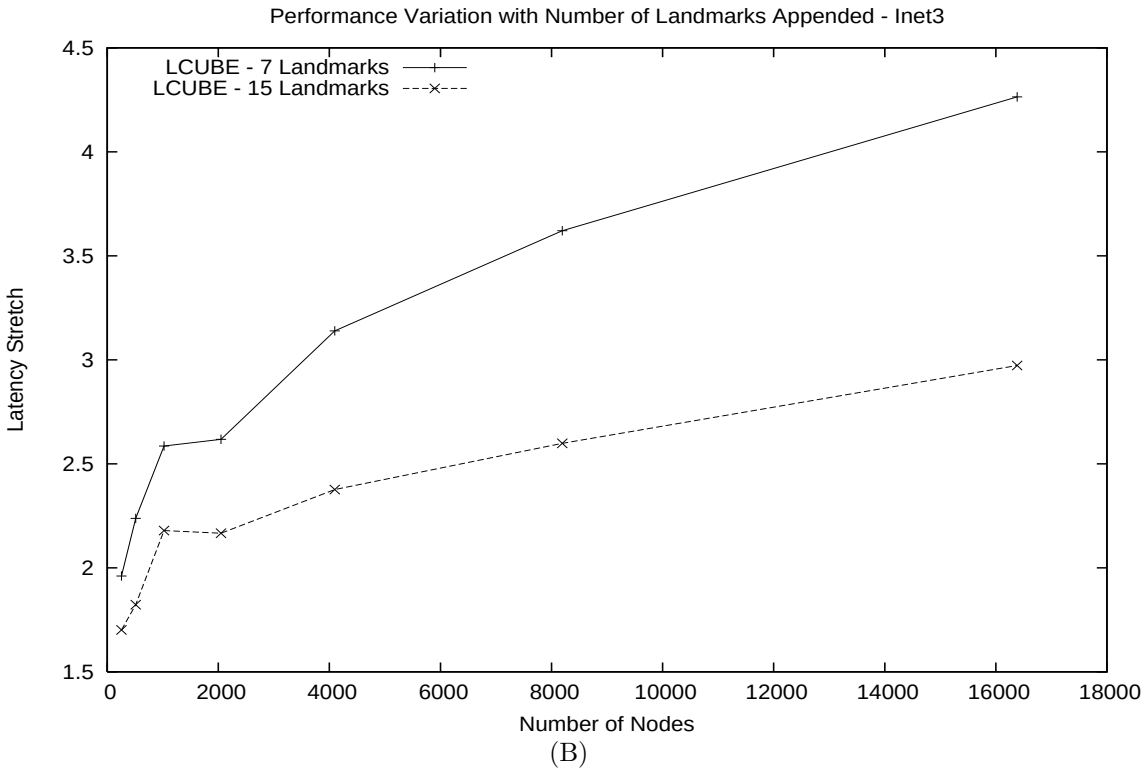
Note that the performance improvement in LCUBE compared to the RCUBE is not much in the Inet3 topology (Figure 7.4(B)), compared to that seen in the GT-ITM topology (Figure 7.4(A)).

7.4 Node Degree Distribution

We present here the node degree distribution for a system with $n = 30$, $\lambda = 1$, and $\mu = 15$. This system can scale up to $30 * 2^{15}$ which is approximately 100K nodes. The Figure 7.5 illustrates the results of this experiment. The topology used does not make much difference in the degree distribution, and hence we provide results for GT-ITM only. We have added the distribution of the 20 dimension RCAN and the RCUBE for comparison. We see that the RCAN maintains less state than both of the Hypercube structures. On the other hand, both of the Hypercubes have pretty much the same node degree distribution, although here again the randomized version has slightly less state.

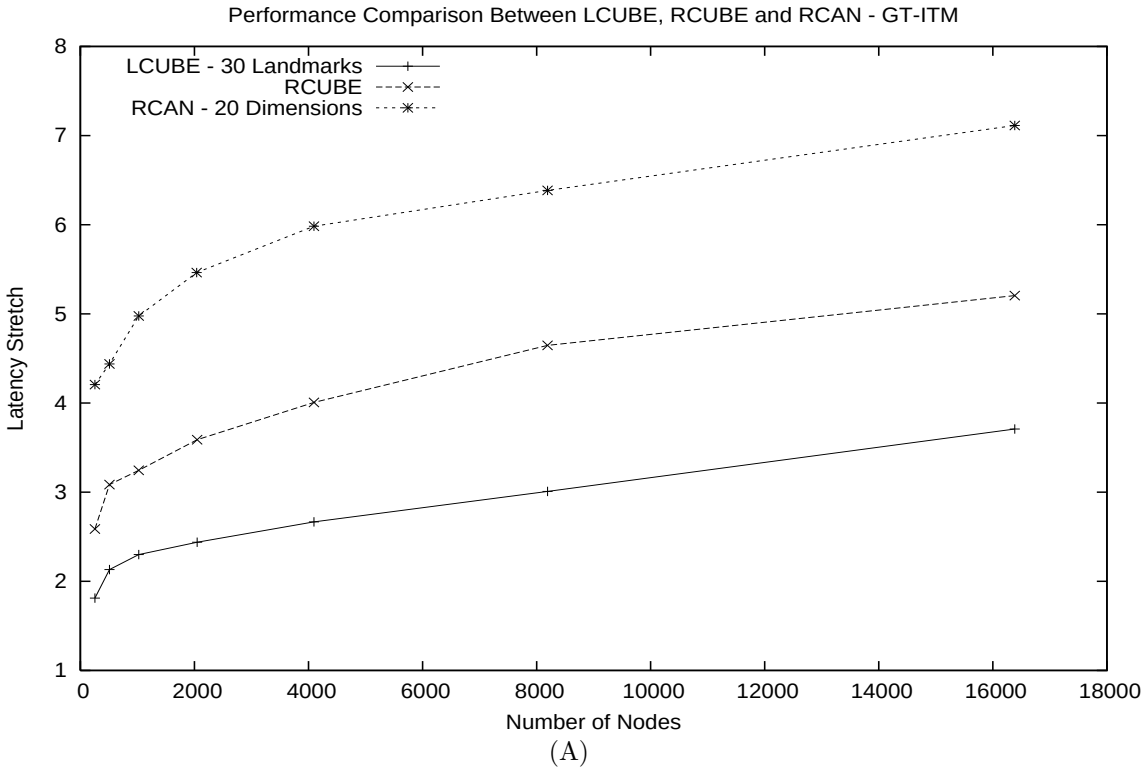


(A)

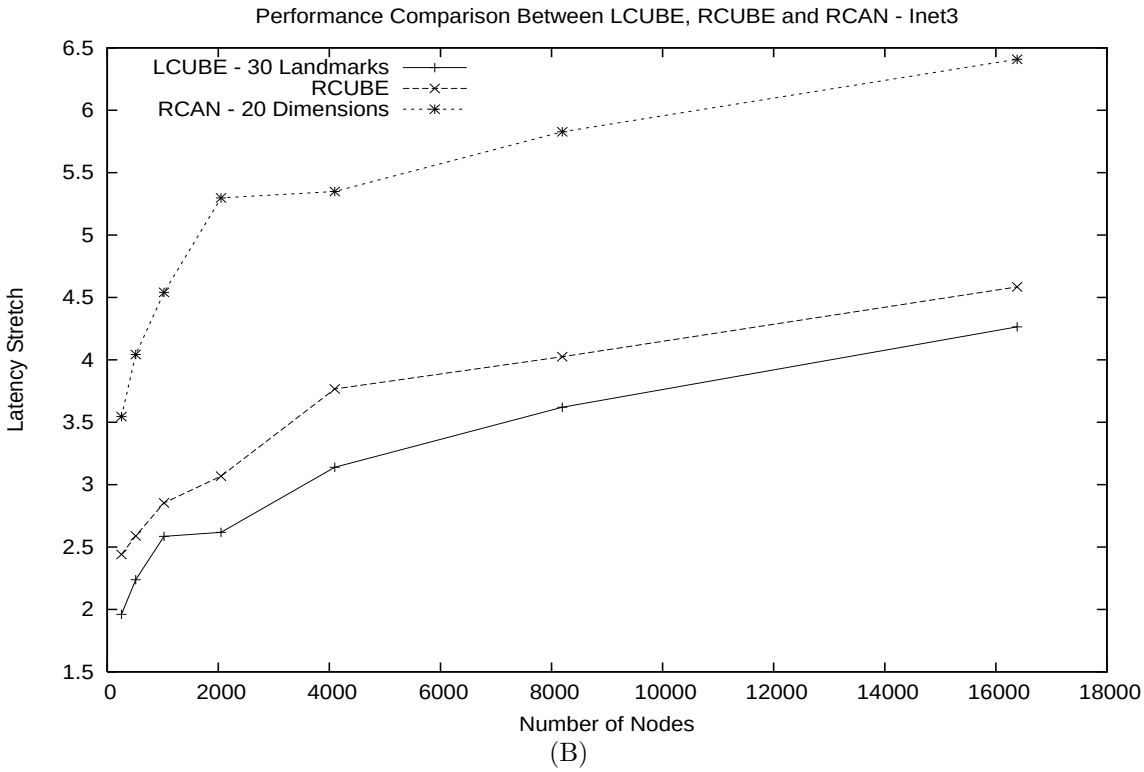


(B)

Figure 7.3: Variation of latency stretch with number of landmarks appended for LCUBE on GT-ITM (A) and Inet3 (B)



(A)



(B)

Figure 7.4: Latency stretch for LCUBE, RCUBE and RCAN on GT-ITM (A) and Inet3 (B)

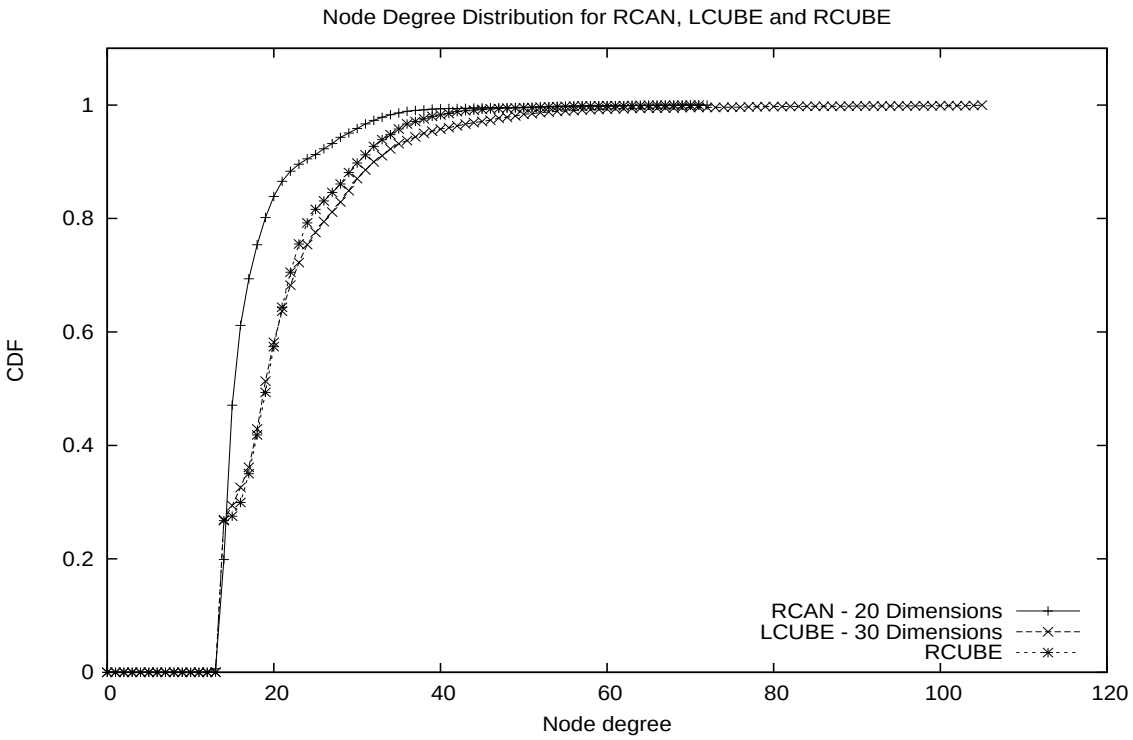


Figure 7.5: Node degree distribution for RCAN, RCUBE and LCUBE

Chapter 8

Conclusion

We presented two techniques for assignment of Proximity Induced Labels to nodes in P2P systems, hence making them topology aware. The first technique, Laminar Binning, works for the 2-d CAN, requiring minimal changes to the existing CAN algorithms and also preserving the desirable properties like load balancing. The second technique, Hypercube Binning, provides a more complex solution for Hypercube networks, requiring more state at each node for improved routing performance at the cost of load balancing.

Hence the Laminar Binning is suitable for pure P2P applications where each and every peer node is assumed to be equal in sharing responsibility and services, with some degree of topology awareness, e.g. Content Distribution Networks, Distributed File Servers/Mirrors on the Internet, etc. On the other hand, the Hypercube P2P system is suitable for applications where a small fraction of nodes are willing to share more responsibility than others, and end-to-end latency is required to be as minimal as possible, e.g. Application Layer Multicast.

One could also think of extending the hypercube labeling to existing P2P structures which use binary labels, e.g. Kademlia. Kademlia is part of a family of XOR networks, where the distance metric between nodes is the hamming distance between their binary labels. This suggests a natural mapping between the node labels in

the Hypercube and the ones in Kademlia. The idea of embedding locality information from landmarks into labels of existing P2P structures in general seems to be a promising area for future research.

Lastly, our Hypercube overlay can be used to achieve approximate PNS in traditional structured P2P systems. In [9], the authors experimentally showed that the ideal PNS, where a node joining the system fills its routing table with the closest among all the candidate nodes, gives almost the same end-to-end latency as the underlying IP network. Of course, such an ideal PNS is currently not practicable since the set of such candidate neighbors is usually very large, and so selecting the closest among them requires too many measurements. We have shown experimentally that the nodes on our Hypercube overlay show a close congruence between their physical topology and the overlay topology. A node on our system can perform an expanding ring search and find all the closest nodes within a few hops. These nodes could be used as an approximation for the PNS. Hence we can combine the benefits of the structured (but randomized) P2P system like load balancing and robustness to correlated failures with the locality awareness provided by our Hypercube overlay.

Bibliography

[1] Markus Aderhold and James Slack. Embedding balanced binary trees in the hypercube.

[2] S. Banerjee, C. Kommareddy, and B. Bhattacharjee. Scalable peer finding on the internet.

[3] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002. To appear.

[4] Woei-Kae K. Chen and M. F. M. Stallmann. On embedding binary trees into hypercubes. *Journal of Parallel and Distributed Computing*, 24(2):132–138, 1995.

[5] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[6] J. Fakcharoenphol, S. Rao, and K. Tawlar. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. In *Proceedings of the Thirty-fifth ACM Symposium on Theory of Computing*, San Diego, CA, 2003.

[7] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[8] P. Francis, S. Jamin, Y. Jin, D. Raz, Y. Shavitt, and L.Zhang. IDMaps: A

Global Internet Host Distance Estimation Service. In *IEEE/ACM Transactions on Networking*, October 2001.

[9] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proceedings of ACM SIGCOMM (2004)*, August 2003.

[10] I. Havel. On hamiltonian circuits and spanning trees of hypercubes. *Casopis. Pest. Mat.*, 109:145–152, 1984.

[11] Pete Keleher, Samrat Bhattacharjee, and Bujor Silaghi. Are virtualized overlay networks too much of a good thing? In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, March 2002.

[12] Jorg Liebeherr and Tyler K. Beam. Hypercast: A protocol for maintaining multicast group members in a logical hypercube topology. In *Networked Group Communication*, pages 72–89, 1999.

[13] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM Press, 2002.

[14] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric. 2002.

[15] E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-based Approaches. In *INFOCOM '02*, New York, NY, June 2002.

[16] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for Scalable Distributed Location. In *Second International Conference on Peer-to-Peer Systems IPTPS '03*, Berkeley, CA, February 2003.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM '01*, San Diego, CA, August 2001.

[18] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware Overlay Construction and Server Selection. In *Proceedings of IEEE INFOCOM '02*, New York, NY, June 2002.

[19] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233, 2001.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[21] Bujor Silaghi, Bobby Bhattacharjee, and Pete Keleher. Query routing in the terradir distributed directory. In *SPIE ITCOM'02*, August 2002.

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. balarkrishnan. Chord: A Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM '01*, San Diego, CA, August 2001.

[23] A. Wagner and D. G. Corneil. Embedding trees in a hypercube is np-complete. *SIAM J. Comput.*, 19(3):570–590, 1990.

[24] Marcel Waldvogel and Roberto Rinaldi. Efficient topology-aware overlay network. *SIGCOMM Comput. Commun. Rev.*, 33(1):101–106, 2003.

[25] Jared Winick and Sugih Jamin. Inet-3.0: Internet topology generator.

[26] Zhichen Xu, Chunqiang Tang, and Zheng Zhang. Building topology-aware overlays using global soft-state. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 500. IEEE Computer Society, 2003.

[27] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, volume 2, pages 594–602, San Francisco, CA, March 1996. IEEE.

[28] Amgad Zeitoun, Chen-Nee Chuah, Supratik Bhattacharyya, and Christophe Diot. An as-level study of internet path delay characteristics.

[29] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.

[30] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20. ACM Press, 2001.