Techniques for Efficient Routing and Load Balancing in Content-Addressable Networks *

Ozgur D. Sahin    Divyakant Agrawal    Amr El Abbadi
Department of Computer Science
University of California at Santa Barbara
{odsahin, agrawal, amr}@cs.ucsb.edu

Abstract

As a Distributed Hash Table (DHT), a Content Addressable Network (CAN) provides efficient routing and object location in a decentralized manner while offering fault tolerance and dynamic peer operations. However, as opposed to other DHTs that use a flat ID space, CAN uses a multi-dimensional logical space. DHTs usually require $O(\log N)$ routing information per peer and provide routing in $O(\log N)$ hops, where $N$ is the number of peers in the system. In CAN, on the other hand, each peer keeps only constant amount of routing information and the routing takes $O(d N^{1/d})$ hops, where $d$ is the dimensionality of the logical space. Hence the routing performance of CAN is worse than other DHTs especially when $d$ is small. In this paper, we describe and evaluate several schemes for efficient routing in CAN by keeping additional routing information at the peers. Furthermore, due to the underlying multi-dimensional ID space, CAN is used by applications that require content-based mapping of data objects onto the ID space. Since uniform hashing is not used, such mappings introduce skewed object distributions among the peers. Thus we also describe load balancing schemes for CAN and investigate their efficiency.

1 Introduction

Distributed Hash Tables (DHTs) [13, 16, 14, 19] have been shown to be an effective way of supporting exact key lookups in a P2P environment. They define a logical structure in the overlay network. Peers participating in the overlay are responsible for portions of the logical structure and keep information about other peers at deterministic relative locations in the logical structures, e.g., neighbors in the logical space [13] or peers at exponentially increasing distances in the ID space [16]. The peers are assigned portions of the logical structure by defining some mapping from their identifiers into the logical structure. Similarly, data objects are mapped into the logical structure using predefined mappings, such as hash functions. Thus each data object is dynamically assigned to a peer in the system. Queries for data objects are mapped into the logical structure using the same mapping and the logical structure of the overlay is utilized for routing the query to the destination peer which holds either the requested object or the corresponding index information.

DHTs support efficient routing in the system. They usually organize the peers into a graph structure based on flat peer IDs. They require $O(\log N)$ routing information at each peer and route messages in $O(\log N)$ hops [16, 14, 19], where $N$ is the number of peers in the system. CAN (Content-Addressable Network) [13] distinguishes itself from those DHTs in several aspects. It uses a multi-dimensional logical space, as opposed to a single dimensional ID space. It requires only constant ($O(d)$) amount of routing information maintained at each peer and routes messages in $O(dN^{1/d})$ hops, where $d$ is the dimensionality of the logical space. The sublinear routing cost of CAN is worse than the logarithmic cost offered by other DHTs, especially when $d$ is small. Routing in CAN is accomplished by passing messages just between the immediate neighbors in the logical space, thus increasing the overall routing cost. In this paper, we discuss several schemes that allow peers to keep pointers to other peers far away in the logical space in addition to their immediate neighbors. These pointers greatly improve the routing performance by enabling direct connections among distant points in the space. The improvement is much more significant for low CAN dimensions where there are only a few immediate neighbors available for routing.

Due to the underlying multi-dimensional logical space, CAN is used by some applications that support complex queries such as content-based lookup [17] and

---

*This research was supported in parts by NSF grants CNF 04-23336, 11S 02-23022, and 11S 02-20152.
event dissemination [7]. Instead of hashing data items, these applications map them directly to the CAN space based on their attribute values or features. Such mappings however result in a non-uniform distribution of data in the system, introducing load imbalance among the peers. Thus we also describe schemes for balancing the storage load in CAN. These schemes allow peers to consider the loads of other peers during join and constantly monitor the load distribution in the system.

The rest of the paper is organized as follows. CAN is described in Section 2. In Section 3 we discuss schemes for efficient routing in CAN. Section 4 describes load balancing schemes for CAN. The experimental evaluation of the proposed schemes is presented in Section 5, and Section 6 concludes the paper.

2 Content-Addressable Network

CAN [13] uses a $d$-dimensional logical structure for the overlay. The logical space is partitioned among the peers in the form of $d$-dimensional axes aligned hyper-rectangular partitions called zones. Each peer is responsible for one zone. Data objects are hashed into the logical $d$-space and an object is indexed by the peer that is responsible for the zone containing the hash point. Figure 1 shows an example 2-dimensional logical space that is partitioned among 9 peers. A data object that is hashed to point $p$ in this case will be indexed by peer $P_8$ since $p$ lies in $P_8$’s zone.

Routing and Lookup: Each peer keeps routing information about the neighboring zones (coordinates of the zone and IP address of the owner peer). When a lookup query for an object is issued at some peer, the object is hashed to its corresponding point and the query is routed toward that point. Each intermediate peer then forwards the query to the neighbor closest to the destination until the peer responsible for the destination point is reached. That peer contains the requested index information, which can be used to locate the data object. In Figure 1, for example, peer $P_2$ keeps routing information about its 4 neighbors: $P_1$, $P_6$, $P_3$, and $P_5$. The arrows in the figure show how a message initiated at peer $P_2$ and destined for point $p$ is routed in the system via neighbor links.

On the average, each peer keeps information about $O(d)$ neighbors and a routing path involves $O(dN^{1/d})$ overlay hops, where $d$ is the dimensionality of the space and $N$ is the number of peers in the system.

Dynamic Peer Operations: CAN is a dynamic system where peers can constantly join or leave the system at will. When a new peer wants to join the system, it sends a join request for a randomly selected point $R$ in the logical space via a peer already in the system. The peer responsible for $R$ then splits its zone into two and hands over one half to the new peer along with the objects lying in that half. The split dimension is selected in a round-robin manner. The new peer learns about its neighbors from the previous owner of its zone. The neighbors affected by this split are then notified so that they update their routing information. When leaving the system, a peer hands over its zone to one of its neighbors. Peers periodically send their zone coordinates and neighbor lists to their neighbors. These heartbeat messages allow peers to learn about changes quickly and also detect peer failures. The failure of a peer is detected by its neighbors as they do not receive any more heartbeat messages. Then one of the neighbors takes over the zone of the failed peer. A background zone-reassignment algorithm tries to avoid fragmentation of the space by assigning a single zone to each peer.

3 Routing Improvements

In CAN, each peer only keeps the routing information about its neighbors in the logical space. Thus the amount of routing state per node is constant and independent of the number of peers in the system. However, especially when $d$ is low, such as 2 or 3, the routing is inefficient compared to other DHTs that provide loga-
rithmic routing performance. Figure 2 shows the routing cost in CAN and Chord [16], each with 16K peers, for different CAN dimensions. Compared to Chord, the routing performance of CAN is quite poor in 2 and 3 dimensions and only becomes comparable after 4 dimensions. The main reason for inefficient routing in low dimensions is that a message can only be routed to a neighbor at each intermediate hop and there are only a few neighbors in low dimensions. As opposed to Chord’s finger pointers to distant IDs, there is no way of taking ‘big jumps’ in the logical space in CAN. In this section, we will discuss some schemes that allow peers to keep routing information (zone coordinates and the IP address) about distant peers in the CAN space in addition to their immediate neighbors. In the rest of the paper, we will refer to those distant neighbors as Long Distance Pointers (LDPs). LDPs are kept in addition to the CAN neighbors for efficient routing at the expense of maintaining more neighbors. A peer routing a message now considers both its neighbors and LDPs, and selects the peer whose zone is closest to the destination as the next hop. Below we describe two different schemes for selecting LDPs. We assume that a peer keeps $k$ LDPs.

3.1 Random Pointers

The naive way of selecting the LDPs is to pick them randomly. In this scheme (random), a peer routes discover messages to $k$ random points in the logical space. Corresponding peers responsible for those points reply with their zone coordinates. The initiating peer then stores the zone coordinates and IP addresses of those peers as its LDPs. Figure 3(a) shows an example where peer $P$ maintains LDPs to 4 peers that are responsible for the selected random points.

3.2 Subspace Pointers

A peer using the random scheme might not have any pointers to certain regions of the logical space since the pointers are selected randomly. For instance, peer $P$ in Figure 3(a) has no pointer to the top left quadrant of the logical space. In order to provide a better coverage of the logical space with LDPs, the subspace scheme partitions the logical space into $k$ equal-sized subregions and selects a random point from each region. Corresponding peers are then used as LDPs. Note that this scheme does not require any global information since each peer can locally determine the corresponding subspaces based on the number of LDPs it wants to maintain. For example, peer $P$ in Figure 3(b) maintains four pointers and selects one from each quadrant.

3.3 Discussion

LDPs represent a trade-off between routing efficiency and maintenance cost. With more LDPs, a peer can route messages more efficiently, but it has to maintain more routing information. The LDP schemes discussed above are quite flexible since a peer can locally decide the number of LDPs it keeps and how to select them, depending on its needs and resources. An analytical study on LDPs in the context of small-world phenomenon is given by Kleinberg [11].

One concern with LDPs is how to keep them up-to-date as peers join and leave the system. Note that the correct routing of messages is guaranteed by the CAN neighbors and LDPs are only required for more efficient routing. Thus LDPs can be maintained in a ‘lazy’ manner, i.e., they are updated only when it is necessary. When a peer $P$ selects an LDP for routing, it tries to contact the corresponding peer. If the peer is not reachable, then $P$ marks that LDP as dead and considers other neighbors and LDPs for routing the message. Later $P$ sends discover messages to update the LDPs that are marked dead. Thus an LDP is updated only after it was selected for routing and found to be unreachable. As a result, LDPs are not directly affected by peer joins and departures.

Other techniques for efficient routing can be employed in addition to those outlined in this section. For instance, peers can cache the recipients of their messages for efficiently routing future messages toward the same ID. In this case, if the peer responsible for an ID is already cached, then a message going to that ID can be sent directly to the cached peer bypassing the overlay routing. This would greatly improve the routing performance if the lookup requests in the system is skewed, i.e., there is a large number of requests for a small set of popular keys. As a future work, we want to design additional LDP schemes that are adaptive to the routing patterns of the peers. For example, a peer might dynamically update its LDPs based on the locations it frequently routes to or the number of hops taken by its messages.

Other schemes have been proposed for efficient routing in CAN. A hierarchical scheme, eCAN [18], pro-
poses to maintain neighbor pointers at different levels of the logical space. It shows that it is possible to obtain logarithmic routing performance by keeping logarithmic routing information at each peer in a CAN system. Similarly, in order to support range queries, MURK [6] partitions a multi-dimensional data space into hyper-rectangles in a manner similar to kd-trees. The partitioning scheme is very similar to CAN with two main differences. First, it does not hash data items but maps them directly onto the data space based on their attribute values. Second, the zone splitting does not have to be at the middle coordinate. MURK uses two schemes for efficient routing in the logical space. The first scheme maintains pointers to random peers in the system, similar to the random scheme discussed in Section 3.1. In the second scheme, each zone in the logical space is mapped to a value by applying a space filling curve on the zone center. A skip graph is built on these values so that each peer keeps pointers to other peers at exponentially increasing distances from it.

4 Load Balancing

An important issue for DHTs is the balancing of load among the peers so that no peer is burdened with excess load. CAN relies on uniform hashing for load balancing. Since both the peers and data items are hashed onto the logical space uniformly, the storage load on any peer is expected to be similar.

Some recent research efforts aim at extending the exact lookup functionality of CAN by providing more complex querying facilities such as content-based similarity search [17], range query caching [15], distributed relational query processing [9], event dissemination [7], and range query support [1, 6]. These extensions usually construct a CAN space whose dimensions correspond to the attributes of the data items shared in the system. For example, pSearch [17] uses the feature vectors of the documents as coordinates in a CAN space in order to support semantic retrieval. Similarly, Meghdoot [7] implements P2P content-based publish/subscribe by distributing events and subscriptions based on their attribute values in a CAN-like logical space. The data items in these systems are not hashed but distributed based on their semantics or attribute values. As a result, the data distribution is not guaranteed to be uniform any more. Thus these systems require explicit load balancing techniques to deal with the skewed data distribution.

Existing load balancing techniques for DHTs [12, 3, 10, 5, 4] usually rely on the hashing of data items or require peers to hand over their partitions to their neighbors. Unfortunately these techniques are not directly applicable to semantic-based CAN systems since the data items are not hashed and it is not always possible to find a neighbor in the CAN space that can be used for merging zones. In this section, we discuss and evaluate some schemes for load balancing in CAN.

4.1 Multiple Join Points

A new peer joining CAN contacts a random peer in the system and takes over half of its zone. A naive extension to the original join protocol is to allow peers to contact multiple peers when joining and select one based on their load. In this scheme (Multiple Join), a new peer randomly contacts multiple existing peers, gets load information from those, and then splits with the peer that has the highest load.

4.2 Forwarded Join

When a peer randomly selects the peer(s) to contact, it might not contact any of the heavily loaded peers. Instead, peers in the system might keep load information about some other peers and forward incoming join requests toward more loaded peers. In this scheme (Forwarded Join), when a new peer $P_n$ joins the system, it sends a join request to a randomly selected peer $P_e$ already in the system. $P_e$ then checks the load values of other peers it knows of to find a peer more loaded than itself. If no such peer exists, $P_e$ processes the join request and hands over half of its zone to $P_n$. Otherwise, it forwards the join request to the peer with the highest load. Thus the join request will reach a maxima, i.e., a peer whose load is greater than the load of any other peer it knows of, and $P_n$ will split the zone of that peer.

The load values can be disseminated in different ways in the system. For example in [7], each peer keeps the load information for its neighbors and a small list of globally loaded peers, and periodically exchanges this information with its neighbors. These exchange messages can be piggybacked in the heartbeat messages used by the underlying CAN system. Another possibility is to maintain an index on the load values, such as a skip graph [2, 8]. This approach allows peers to locate the peer with the highest load in the system at the expense of maintaining a separate index [5].

4.3 Multiple Zones

The two schemes discussed so far can only be executed during peer joins, and therefore they might not be able to balance the load in some circumstances. For example they will not work if there are no new peers joining the system, but a load imbalance arises as a result of data insertions and deletions. A more dynamic approach is to have existing peers periodically check the load of other peers and initiate a ‘load balance’ action if necessary. With Multiple Zones, peers keep load information about some other peers and join
requests are forwarded to loaded peers (same as Forwarded Join). Additionally, each peer \( P_i \) in the system periodically compares its load with that of the most loaded peer \( P_h \) it knows of. If the load ratio is smaller than a threshold value \( T_h \), then \( P_i \) hands over its zone(s) to its neighbor(s), and splits with peer \( P_h \). When handing over a zone, \( P_i \) selects the peer with the smallest load among the neighbors of that zone. Then \( P_h \) splits its zone(s) with \( P_i \) such that the load is evenly divided between the two peers. For this purpose, \( P_h \) transfers some of its zone(s) to \( P_i \), and might split one of the zones.

With this scheme, peers may need to maintain multiple zones, however the ratio of the smallest load to the highest load in the system is bounded by the threshold \( T_h \). To avoid the fragmentation of the space, each peer may check to see if its zone can be merged with a neighboring zone. If such a zone exists, the two peers might merge their zones. They might also exchange their other zones to keep their loads within the threshold. We will investigate the effect of creating multiple zones per peer on the routing and maintenance cost in Section 5.2.

4.4 Discussion

Several schemes have been proposed for guaranteeing load balance in semantic-based DHTs [5, 10, 4]. However, these designs require peers to hand over their partitions to their neighbors and to split their partitions in a way that results in even partitioning of the load. Thus they are not directly applicable to CAN. For better load balancing, [12] proposes for each physical peer to create multiple virtual peers in the system. The Multiple Zones scheme can therefore be considered as a combination of these two approaches. pSearch [17] proposes content-aware joins where a new peer randomly selects a document it is going to share, and uses its feature vector to decide where to join in the CAN space. As a result, there will be more peers in the regions that have more documents.

In this section we only focused on the balancing of the storage load. Another important problem to consider is the processing load, which is caused by routing messages and answering lookup requests. The processing load imbalance can be mitigated with some of the design improvements proposed by CAN [13]. For example, the routing load on a peer can be reduced by assigning its zone to multiple peers (Overloading Zones). Similarly, to reduce the load caused by incoming lookup requests for popular data items, peers can cache the data items they recently accessed (Caching) and loaded peers can replicate such popular items at their neighbors (Replication). In Meghdoot [7], a new peer considers both the processing and storage load to decide whether to split or replicate a zone. Splitting is useful for reducing the storage load whereas zone replication helps reducing the processing load.

The join algorithm of CAN splits the zones at the middle coordinates in order to keep the routing cost and number of neighbors low. This might however reduce the effectiveness of the load balancing schemes since sometimes the data items might be clustered at a small region of the zone and splitting the zone does not reduce the storage load. Thus some applications built on top of CAN allow peers to split their zones at the median coordinate of the stored data items [15, 7, 6]. Such splitting might increase the average number of routing hops and neighbors in the system, but provides even split of the stored data items. We leave the integration of the processing load and non-regular partitioning into the load balancing schemes as a future work.

5 Experimental Results

We evaluate the routing and load balancing schemes discussed using a CAN simulator implemented in Java. Unless stated otherwise, there were 16K peers in the system, and the dimensionality of CAN was set to 2 by default. The measurements were taken after all peers joined the system. We did not simulate any peer failures or departures in the experiments.

5.1 Evaluation of Routing Improvements

In this section we evaluate the two routing optimization schemes (random and subspace) presented in Section 3. We compare their performances with that of the original CAN (the load balancing schemes are not used). We measure the average number of hops visited for routing a message in the system by varying three parameters: the dimensionality of CAN, the number of LDPs, and the number of peers. The default number of LDPs is set to 16 since it is a power of 2 and is close to \( \log N \) (Note that \( \log N = 14 \) in the default setting). When deciding on the subspaces, a peer keeps halving the sub-intervals along each dimension cyclically until the desired number of subspaces is achieved. For instance in 3 dimensions, the \( x \) dimension is split into 4 equal intervals, whereas the \( y \) and \( z \) dimensions have only 2 intervals each (a total of \( 4 \times 2 \times 2 = 16 \) subspaces). The results presented here are averaged over \( 10^6 \) messages that were routed after all peers joined the system. Each of these messages was initiated at a random peer and the destination was set to a point selected uniformly at random from the logical space. Since the results for the two schemes are similar, we present them in tabular format. The results for the subspace scheme and CAN are also shown graphically in Figure 4.
We first measured the routing improvements provided by each scheme over the original CAN for different dimensions. Table 1 shows the average routing path length, i.e., the number of peers visited during routing, in each case for dimensions 2 to 7 (also see Figure 4(a)). The results indicate that the routing performance of CAN is low especially for low dimensions, and each of the discussed LDP schemes can greatly improve the performance. The improvement is much more significant in low dimensions such as 2 and 3. That is because the average number of neighbors in low dimensions is very small and thus peers have only a few peers for forwarding the messages. In 2 dimensions for example, LDPs provide over 80% improvement in routing by reducing the routing cost from 50 hops to around 9.5 hops. The routing is more efficient with LDPs for each dimension, but the amount of improvement gets smaller as the dimensionality increases because CAN is already efficient in high dimensions. The subspace scheme provides slightly better performance than the random scheme due to its better space coverage.

Table 1. Routing Cost vs Dimensionality

<table>
<thead>
<tr>
<th>Dim.</th>
<th>CAN</th>
<th>Random</th>
<th>Subspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>50.22</td>
<td>9.60</td>
<td>9.41</td>
</tr>
<tr>
<td>3</td>
<td>15.60</td>
<td>6.46</td>
<td>6.39</td>
</tr>
<tr>
<td>4</td>
<td>9.68</td>
<td>5.42</td>
<td>5.37</td>
</tr>
<tr>
<td>5</td>
<td>7.71</td>
<td>5.00</td>
<td>4.96</td>
</tr>
<tr>
<td>6</td>
<td>6.77</td>
<td>4.84</td>
<td>4.81</td>
</tr>
<tr>
<td>7</td>
<td>6.31</td>
<td>4.74</td>
<td>4.72</td>
</tr>
</tbody>
</table>

We also measured the average number of CAN neighbors in each dimension. Table 2 shows that the number of neighbors is usually very small. Considering the significant improvements in the routing performance and the lightweight maintenance of LDPs compared to regular CAN neighbors, we believe that it is acceptable for the peers to maintain LDPs for more efficient routing.

Table 2. Number of Neighbors in CAN

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Nghbrs</td>
<td>4.6</td>
<td>7.2</td>
<td>9.9</td>
<td>12.5</td>
<td>15.1</td>
<td>17.7</td>
</tr>
</tbody>
</table>

We also measured the average number of LDPs visited during routing. Table 3 shows the routing cost for different number of LDPs in 2 dimensions. Note that the corresponding cost for CAN, which has no LDPs, is 50.22. Therefore, the routing cost is reduced more than 50% by keeping just one LDP. As the number of LDPs increases, the routing cost reduces, but the reduction gets smaller with more LDPs. Since keeping more LDPs requires additional maintenance cost for the peer, keeping between 8 to 32 LDPs seems reasonable considering the corresponding performance gains (Figure 4(b)).

Table 3. Routing Cost vs Number of LDPs

<table>
<thead>
<tr>
<th>LDP #</th>
<th>Random</th>
<th>Subspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.12</td>
<td>22.12</td>
</tr>
<tr>
<td>2</td>
<td>18.02</td>
<td>17.77</td>
</tr>
<tr>
<td>4</td>
<td>14.58</td>
<td>14.45</td>
</tr>
<tr>
<td>8</td>
<td>11.83</td>
<td>11.64</td>
</tr>
<tr>
<td>16</td>
<td>9.60</td>
<td>9.41</td>
</tr>
<tr>
<td>32</td>
<td>7.81</td>
<td>7.64</td>
</tr>
<tr>
<td>64</td>
<td>6.35</td>
<td>6.16</td>
</tr>
<tr>
<td>128</td>
<td>5.10</td>
<td>4.93</td>
</tr>
</tbody>
</table>

Finally we investigated the routing performance for varying number of peers. The results are shown in Table 4 and Figure 4(c). For all three cases, the routing cost increases with the increasing number of peers. However, the cost is much smaller and increases slowly with LDPs. For example, when the number of peers changes from 4K to 64K, the routing cost for CAN increases from 25.60 to 100.23 (~4x increase), whereas it increases from 6.08 to 14.68 in the case of subspace pointers (~2.5x increase).

Table 4. Routing Cost vs Number of Peers

<table>
<thead>
<tr>
<th>Peer #</th>
<th>CAN</th>
<th>Random</th>
<th>Subspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>13.15</td>
<td>4.10</td>
<td>3.95</td>
</tr>
<tr>
<td>2K</td>
<td>18.62</td>
<td>5.06</td>
<td>4.93</td>
</tr>
<tr>
<td>4K</td>
<td>25.60</td>
<td>6.18</td>
<td>6.08</td>
</tr>
<tr>
<td>8K</td>
<td>36.56</td>
<td>7.80</td>
<td>7.55</td>
</tr>
<tr>
<td>16K</td>
<td>50.22</td>
<td>9.60</td>
<td>9.41</td>
</tr>
<tr>
<td>32K</td>
<td>71.47</td>
<td>11.95</td>
<td>11.75</td>
</tr>
<tr>
<td>64K</td>
<td>100.23</td>
<td>14.81</td>
<td>14.68</td>
</tr>
</tbody>
</table>
5.2 Evaluation of Load Balancing Schemes

In this section, we evaluate the load balancing techniques presented in Section 4. For these experiments, we used a synthetic skewed data distribution where the coordinates of the data points along each dimension are selected as follows. We generated values from a standard normal distribution (mean 0, standard deviation 1) and only used the ones that were in the [-2, 2] interval. We mapped those values to the [0, 1] interval by taking their absolute values and dividing by 2. Each peer inserted 10 data items into the system upon its join, and after all peers joined the system 2 more data items per peer were inserted. We used the following default parameters for the load balancing schemes. For Multiple Join, new peers contact 4 random peers. In the case of Forwarded Join and Multiple Zones, we assume that a separate index is maintained on the load values, thus a new peer always splits the zone of the most loaded peer. The load ratio threshold, $T_L$, for Multiple Zones is set to 0.4 so that a peer initiates a load balance action (hands over its zone and splits the zone of the loaded peer) if the ratio of its load to that of the most loaded peer is less than 0.4. For the dynamic load exchange in Multiple Zones, after all peers joined the system, each peer checks once whether it should perform load exchange.

Figure 5 shows the cumulative load distribution in the system for each scheme for a 2-d CAN system with 16K peers. The peers are sorted in decreasing order of load, i.e., the number of data items assigned. The $x$ axis shows the percentage of the top peers from the sorted list whereas the corresponding $y$ value is the percentage of the data items assigned to those peers. Note that for the optimal case where each peer has equal load, the corresponding plot would lie on the $y = x$ line. As the graph shows, all three load balancing schemes significantly reduce the load imbalance in the system. For example in CAN, top 20% of the peers are responsible for 55.4% of the data items. After applying the load balancing schemes, that value drops to 34.3% for Multiple Join, 27.9% for Forwarded Join, and 26.6% for Multiple Zones. Forwarded Join and Multiple Zones are more effective than Multiple Join since they use a separate index for accurately locating the loaded peers. However note that keeping such an index introduces additional maintenance cost since the index should be updated as the loads of the peers change. Multiple Zones performs slightly better than Forwarded Join as it also provides dynamic load balancing. Its performance can be further improved by increasing the value of the load ratio threshold, $T_L$. Similarly, Multiple Join performs better when the number of contacted peers increases. We skip the evaluation of those improvements due to space constraints.

Table 5 shows the average number of routing hops and neighbors per zone after implementing each load balancing scheme. The results suggest that as the load balancing scheme gets more effective, the number of neighbors drops and the routing cost increases. The reason for less number of neighbors is the large zones in the sparse regions of the data space. Another reason for less efficient routing in Multiple Zones is the increased number of zones due to the dynamic load balancing operation. In this experiment, no peer had more than one additional zone and the total number of additional zones was 1140. Thus the average number of neighbors per peer was 4.59. One possible routing optimization in this case is that a peer considers the neighbors of all zones it maintains when routing a message. With this improvement, the routing cost for Multiple Zones dropped from 61.39 to 50.32. Note that the routing performance in all three schemes can be improved using the LDP schemes discussed in Section 3. Thus it is possible to achieve both efficient routing and load balancing in CAN. We also evaluated the proposed load balancing schemes for higher dimensions and observed similar trends in terms of load balancing and routing cost. The load distribution for CAN was more skewed in higher dimensions, but the three schemes effectively balanced the load with even greater improvements than 2 dimensions. Figure 6 shows the corresponding cumulative load distribution for 6 dimensions. In this case, there were 1325 additional zones for Multiple Zones.
6 Conclusions

As a DHT, CAN provides efficient routing and lookup performance in a P2P environment. In contrast to other DHTs that use a flat ID space and offer logarithmic routing performance, CAN uses a d-dimensional logical space and provides sublinear routing cost. In this paper, we showed that the routing performance of CAN can be increased significantly by keeping a small number of additional neighbor pointers at the peers. In fact, it achieves performance comparable to logarithmic routing DHTs, namely Chord. We evaluated two schemes, random and subspace, and concluded that selecting the additional pointers from different subregions of the logical space provides better performance than selecting them randomly. We also discussed and evaluated schemes for load balancing in CAN when the data distribution in the system is not uniform. Such skewed distributions appear when the data is distributed based on its semantics or attribute values. With these load balancing schemes, new peers can consider the loads of other peers when choosing the location to join, and existing peers can monitor the load distribution in the system and initiate necessary load exchange operations. Our results show that efficient load balancing can be achieved at the expense of a slightly reduced routing performance.

As future work, we plan to design additional schemes for efficient routing that adapt to the routing patterns of the peers. Additionally, we want to evaluate the performance when both routing and load balancing schemes are used together.

References