# Locality-Aware Randomized Load Balancing Algorithms for DHT Networks

Haiying Shen and Cheng-Zhong Xu
Department of Electrical & Computer Engineering
Wayne State University, Detroit, MI 48202
{shy,czxu}@ece.eng.wayne.edu

## Abstract

*Structured P2P overlay networks based on a consistent hashing function have an aftermath load balance problem that needs to be dealt with. A load balancing method should take into account both proximity and dynamic features of DHTs. Randomized matching between heavily loaded nodes with lightly loaded nodes can deal with the dynamic feature. But current randomized methods are unable to consider physical proximity of the node simultaneously. There are locality-aware methods that rely on an additional logical network to capture the physical locality in load balancing. Due to the cost for network construction and maintenance, these locality-aware algorithms can hardly deal with DHTs with churn. This paper presents a locality-aware randomized load balancing algorithm to deal with both of the proximity and dynamic features of DHTs. We introduce a factor of randomness in the probing process in a range of proximity to deal with the DHT churn. We further improve the randomized load balancing efficiency by d-way probing. Simulation results show the superiority of a locality-aware 2-way randomized load balancing in DHTs, in comparison with other pure random policies and locality-aware sequential algorithms. In DHTs with churn, it performs no worse than the best churn resilient algorithm.*

## 1 Introduction

Over the past years, the immerse popularity of peer-to-peer (P2P) resource sharing services has produced a significant stimulus to content-delivery overlay network research. An important class of the overlay networks is distributed hash tables (DHTs) [7, 11, 8, 10] that map keys to the nodes of a network based on a consistent hashing function. However, consistent hashing [3] produces a bound of $O(\log n)$ imbalance of keys between nodes, where n is the number of nodes in the system. Load balancing algorithm is to avoid load imbalance by distributing application load among the nodes in proportional to node capacities. The design of a load balancing algorithm should take into account DHT proximity feature to minimize load balancing cost and dynamic feature to handle churn — a situation where a great number of nodes join, leave and fail continually and rapidly.

In the past, numerous load balancing algorithms were proposed with different characteristics [11, 6, 2, 16, 4]. However, few of them are able to deal with both the dynamism and proximity. In general, the dynamic feature of DHTs should be dealt with by randomized matching between heavily loaded nodes with lightly loaded nodes. Rao and Godfrey *et al.* [6, 2] proposed randomized load balancing algorithms for dynamic DHTs with churn. The algorithms treat all nodes equally in random probing, without consideration of node proximity information in load balancing. Zhu and Hu presented a proximity-aware algorithm to take into account the node proximity information in load balancing [16]. The algorithm is based on an additional network constructed on top of DHTs. Although the network is self-organized, the load balancing algorithm is hardly applicable to DHTs with churn.

In this paper, we present novel locality-aware randomized (LAR) load balancing algorithms to deal with both the proximity and dynamic features of DHTs. The algorithms take advantage of the proximity information of the DHTs in node probing and distribute application load among the nodes according to their capacities. We introduce a factor of randomness in the probing process in a range of proximity so as to make the load balancing algorithm resilient enough to deal with the dynamic feature of DHTs. We further improve the efficiency of the random probing process by d-way probing. The algorithms are implemented in Cycloid [10], based on a concept of "moving item" for retaining DHTs' network efficiency and scalability. We evaluated the performance of the LAR load balancing algorithms via comprehensive simulations. Simulation results demonstrate the superiority of a locality-aware 2-way randomized load balancing algorithm in DHTs, in comparison with other pure random approaches and locality-aware sequential algorithms. In DHTs with churn, it performs no worse than the best churn resilient algorithm.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative load balancing approaches for structured P2P systems. Section 3 details a load balancing framework. Section 4 presents the LAR load balancing algorithms. Section 5 shows the performance of the approaches in terms of a variety of metrics in DHTs

with and without churn. Section 6 concludes this paper with remarks on possible future work.

## 2 Related Work

Load balance is an aftermath problem in any DHTs based on consistent hashing functions. Karger *et al.* proved that the consistent hashing function in chord [11] leads to a bound of $O(\log n)$ imbalance of keys between the nodes. Stoica *et al.* proposed an abstraction of "virtual servers" for Chord load balancing. This abstraction simplifies the treatment of load balancing problem at the cost of higher space overhead and lookup efficiency compromise. The original concept of "virtual servers" ignored the file size and node heterogeneity. Later on, Rao *et al.* [6] proposed three algorithms to rearrange load based on nodes' different capacities. Their basic idea is to move load from heavy nodes to light nodes so that each node's load does not exceed its capacity. Most recently, Godfrey *et al.* [2] extended this work for dynamic P2P systems. This is, if a node's capacity utilization exceeds a predetermined threshold, its excess virtual servers will be moved to a light one immediately without waiting for next periodic balancing. The algorithms assumes a goal of minimizing the amount of load moved. They neglects the effect of proximity information. With proximity consideration, load transferring and communication are between physically close heavy nodes and light nodes. One of the first work to utilize the proximity information to guide load balancing is due to Zhu *et al.* [16]. The authors suggested to build a K-nary tree (KT) structure on top of a DHT overlay. The KT tree helps to use proximity information to move load between physically close heavy and light nodes. However, the construction and maintenance of KT are costly, especially in churn. Besides, when a parent fails or leaves, the load imbalance of some of its children in the subtree cannot be solved before its recovery. Most recently, Karger and Ruhl [4] proved that the "virtual servers" method could not be guaranteed to handle item distributions in a certain condition. As a remedy, they proposed two schemes with provable features: *moving items* and *moving nodes* to achieve equal load between a pair of nodes, and then a system-wide load balance state.

This paper presents LAR algorithms to take into account proximity information in load balancing and deal with network churn meanwhile and a first implementation of item movement based load balancing algorithms though it is also suitable for "virtual servers".

## 3 Load Balancing Framework

Cycloid is a lookup efficient constant-degree DHT with $n{=}d \cdot 2^d$ nodes, where $d$ is dimension. Each Cycloid node has $O(1)$ neighbors and is represented by a pair of indices $(k, a_{d-1}a_{d-2}\dots a_0)$, where k is a cyclic index and

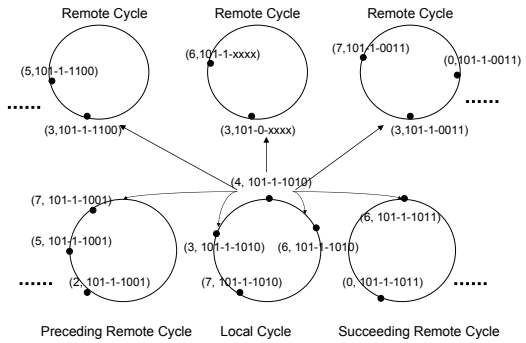



**Figure 1. Cycloid node routing links state.**

$a_{d-1}a_{d-2}.....a_0$ is a cubical index. The cyclic index is an integer, ranging from 0 to $d-1$ and the cubical index is a binary number between 0 and $2^d-1$. The nodes with the same cubical index are ordered by their cyclic index mod $d$ on a *local cycle*. The largest cyclic index node in a local cycle is called the *primary node* of the nodes at the same local cycle. All local cycles are ordered by their cubical index mod $2^d$ on a *large cycle*. Figure 1 shows the routing links of a Cycloid node (4,10111010). They include the node's predecessor and successor in the local cycle, two primary nodes of the preceding and the succeeding remote cycles, one *cubical neighbor* and two *cyclic neighbors*. For more information about Cycloid, please refer to [10].

In the following, we will present a framework for load balancing based on "moving item" on Cycloid. It takes advantage of the Cycloid's topological properties and conduct a load balancing process in two steps: *local load balancing* within a cluster and *global load balancing* between clusters.

A general load balancing approach with consideration of node heterogeneity is to partition nodes into a super node with high capacity and a class of regular nodes with low capacity [14]. Each super node, together with a group of regular nodes, forms a cluster in which the super node operates as a server to the others and all the super nodes operate as equal nodes. Super-peer network strikes a balance between the inherent efficiency of centralization and distribution, and takes advantage of capacity heterogeneity, as well. Since Cycloid consists of cycles with a primary node in each cycle, we build a Cycloid super-peer network by assigning each primary node as a super node in its cycle/cluster. The neighborhood construction mechanism in [12] can be used to construct super-peer networks in other DHTs such as Chord, Pastry, etc.

Let $L_i$ represent the *actual load* of a real server $i$. It is the sum of the load of the items it stores: $L_i = \sum_{k=1}^{m_i} L_{i,k}$, assuming the node has $m_i$ items. Let $C_i$ be the capacity of node $i$ and $T_i$ its pre-defined target load in a percentage of the node capacity. We refer to the node whose actual load is no larger than its target load (*i.e.* $L_i \leq T_i$) as a *light node*; otherwise a heavy node. We define *node utilization* $NU_i$ as the fraction of its target capacity that is used: $NU_i =$

$L_i/T_i$. A *system utilization* is the fraction of the system's total target capacity.

Each node contains a list of data items, labelled as $ID_k$ $(k = 1, 2, \ldots)$. To reduce the workload of a heavy node, the items chosen to transfer are called *excess items*. Each primary node has a pair of donating sorted list (DSL) and starving sorted (SSL) list which store the load information of all nodes in its cluster. A DSL is for light nodes and a SSL is for heavy nodes. The *free capacity* of light node i, $\triangle L_i$, is defined as $\triangle L_i = T_i - L_i$. *Load information* of a heavy node i is expressed as $<L_{i,1}, ID_{i,1}, ip\_addr(i)>$, $<L_{i,k}, ID_{i,k}, ip\_addr(i)>$, $\ldots$, $<L_{i,m}, ID_{i,m}, ip\_addr(i)>$, in which $ip\_addr(i)$ denotes the IP address of node i. Load information of a light node j is expressed as $<\triangle L_j, ip\_addr(j)>$. A SSL is sorted in a descending order of $L_{i,k}$. A DSL is sorted in an ascending order of $\triangle L_j$. Load rearrangement is executed between a pair of DSL and SSL, as shown in Algorithm 1. This scheme guarantees that heavier items have high priorities to be reassigned to a light node, which means faster convergence to a system-wide load balance state.

**Algorithm 1** Primary node performs load rearrangement periodically between a pair of DSL and SSL.

```
1: for each item k in SSL do
2:   for each item j in DSL do
3:     if L_{i,k} <= △L_j then
4:       item k is arranged to be transferred from i to j
5:       if △L_j − L_{i,k} > 0 then
6:         put < (△L_i − L_{i,k}), ip_addr(i) > back to DSL
7:       end if
8:     end if
9:   end for
10: end for
```

We use a centralized method for local load balancing, and a decentralized method for global load balancing. Periodically, each node reports its load information to its primary node. A primary node with nonempty starving list (PNS) first performs local load rearrangement between its DSL and SSL. It then probes other primary nodes' DSLs for global load rearrangement until its SSL becomes empty. This scheme can be extended to perform load rearrangement between one SSL and multiple DSLs for improvement.

# 4 Locality-Aware Randomized Load Balancing Algorithms

The load balancing framework presented in the preceding section facilitates the development of load balancing algorithms with different characteristics. The key difference between the algorithms is, for a PNS, how to choose another primary node for a global load rearrangement between their SSL and DSL. It affects the efficiency and overhead to reach a system-wide load balance state.

## 4.1 D-way Randomized Probing

In a randomized probing policy, each PNS probes other primary nodes randomly for load rearrangement. A simple form is one-way probing, in which a PNS, say node $i$, probes other primary nodes one by one to execute load rearrangement between $SSL_i$ and $DSL_j$, where $j$ is a probed node.

The randomized probing in our load balancing framework is similar to load balancing problem in other contexts: *competitive online load balancing* and *supermarket model*. Competitive online load balancing is to assign each task to a server on-line with the objective of minimizing the maximum load on any server, given a set of servers and a sequence of task arrivals and departures. Azar *et al.* [1] proved that in competitive online load balancing, allowing each task to have two server choices to choose a less loaded server instead of just one choice can exponentially minimize the maximum server load and result in a more balanced load distribution. Supermarket model is to allocate each randomly incoming task modelled as a customer with service requirements, to a processor (or server) with the objective of reducing the time each customer spends in the system. Mitzenmacher *et al.* [5] proved that allowing a task two server choices and to be served at the server with less workload instead of just one choice leads to exponential improvements in the expected execution time of each task. But a poll size larger than two gains much less substantial extra improvement.

The randomized probing in our load balancing framework can be represented by the above models if we regard SSLs as tasks, and DSLs as servers. As to the problem of the same capacity for each server in those models, we treat the condition that different DSLs have different total free capacity in the same way as that those DSLs are already assigned some tasks and with different free capacity left; that is, in the middle way of task assignment with a same very large total target capacity for each of DSLs. We generalize the one-way random probing policy to a d-way probing, denoted by $R_d$ (d$\geq$1), in which d primary nodes are probed at a time, and the primary node with the most total free capacity in its DSL is chosen for load rearrangement. We expect that 2-way probing could achieve a more balanced load distribution with faster speed even in churn, but d ($>$2)-way probing may not result in much additional improvement.

## 4.2 Locality-Aware Probing

One goal of load balancing is to keep each node's utilization below one with minimum overhead and time. Proximity is one of the most important performance factors. We integrate proximity-neighbor selection and topologically-aware overlay construction techniques [13] into Cycloid to build a topology-aware Cycloid. That is, a node selects the routing table entries pointing to the topologically nearest

among all nodes with nodeId in the desired portion of the Id space, and each node Id is represented by its Hilbert number so that physically close primary nodes are close to each other in the large cycle. As a result, the primary nodes of a node's neighbors are closer to the node than randomly chosen primary nodes in the entire network, such that the cost for communication and load movement can be reduced if a primary node contacts its primary node neighbors or primary nodes of its neighbors. There are two methods for locality-aware probing: *randomized* and *sequential* method. In locality-aware randomized probing, each PNS contacts its primary node neighbors or primary nodes of its neighbors. After all neighbors have been tried, if the PNS's SSL is still nonempty, global random probing is started in the entire Id space. In locality-aware sequential probing, denoted by Lseq, each PNS contacts its successor, *Successor(PNS)*. After load rearrangement, if its SSL is still nonempty, *Successor(Successor(PNS))* is tried. This process is repeated, until that SSL becomes empty.

# 5 Performance Evaluation

We designed and implemented a simulator in Java for evaluation of the load balancing algorithms on topology-aware Cycloid. We selected 15 nodes as landmark nodes to generate the landmark vector and a Hilbert number [13] for each node Id. We use two transit-stub topologies generated by GT-ITM [15]: "ts5k-large" and "ts5k-small". "ts5k-large" has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. "ts5k-small" has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. "ts5k-large" is used to represent a situation in which Cycloid overlay consists of nodes from several big stub domains, while "ts5k-small" represents a situation in which Cycloid overlay consists of nodes scattered in the entire Internet and only few nodes from the same edge network join the overlay. To account for the fact that interdomain routes have higher latency, each interdomain hop counts as 3 hops of units of latency while each intradomain hop counts as 1 hop of unit of latency. Table 1 lists the parameters of the simulation and their default values. Pareto distribution reflects real world where there are machines with capacities that vary by different orders of magnitude. We will compare the different load balancing algorithms in Cycloid without churn in terms of the following performance metrics; performance of the LAR in Cycloid with churn will be evaluated in Section 5.4.

(1) *Load movement factor*, defined as the total load transferred due to load balancing divided by the system actual load. It represents load movement cost for load balance.
(2) *Total time of probings*, defined as the time spent for primary node probing assuming that probing one node takes 1 time unit, and probing n nodes simultaneously also takes 1 time unit. It represents the speed of probing phrase in load balancing to achieve a system-wide load balance state.
(3) *Total number of load rearrangements*, defined as the total number of load rearrangement between a pair of SSL and DSL.
(4) *Total probing bandwidth*, defined as the sum of the bandwidth consumed by all probings. A probing's bandwidth is the sum of the bandwidth of all communication, each of which is message size times physical path length of the message travelled. It is assumed that the size of a message asking and replying for information is 1 unit. It represents the traffic burden caused by probings.
(5) *Moved load distribution*, defined as the cummulative distribution function (CDF) of the percentage of moved load versus moving distance. It represents the load movement cost for load balance. The more load moved along the shorter distances, the less load balancing costs.

**Table 1. Simulation settings and algorithm parameters.**

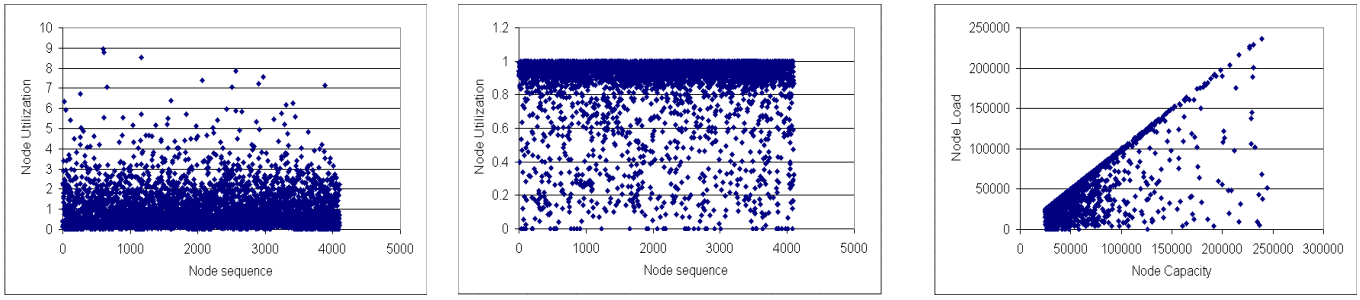
| Environment Parameter | Default value |
|---|---|
| Object arrival location | Uniform over Id space |
| Number of nodes | 4096 |
| Node capacity | Bounded Pareto: shape 2<br>lower bound:2500, upper bound: 2500*10 |
| Number of items | 20480 |
| Existing item load | Bounded Pareto: shape: 2,<br>lower bound: mean item actual load/2<br>upper bound: mean item actual load/2*10 |

Because metrics (2) and (3) are not affected by topology, we will only show results of them in "ts5k-large".

## 5.1 Effectiveness of LAR Algorithms

In this section, we will show the effectiveness of LAR load balancing algorithms. First, we present the impact of LAR algorithm on the alignment of the skews in load distribution and node capacity when the system is fully loaded. From Figure 2(a) and (b), we can see that many nodes are overloaded before load balancing and after load balancing they become light by transferring excess items to light nodes. Figure 2(c) shows the scatterplot of loads according to node capacity. These figures show that the load balancing frame assigns load to nodes based on their capacity with the consideration of node heterogeneity.

We measured the load movement factors due to different load balancing algorithms: one-way random ($R_1$), two-way random ($R_2$), $LAR_1$, $LAR_2$, and Lseq, on systems of utilization from 0.5 to 1, with 0.05 increase in each step. The simulation results showed that the algorithms require the same




(a) Before load balancing (b) After load balancing (c) Utilization of nodes after load balancing

**Figure 2. Effect of load balancing**

amount of load movement in total for load balance. This is in consistent with the observations by Rao, *et al.* [6] that the load moved depends only on distribution of loads, the target to be achieved and not on load balancing algorithms. This result suggests that a better load balancing algorithm should explore how to move the same amount of load along shorter distance to reduce item transfer cost; in another word, how to achieve locality-aware load balancing. In the following, we will examine the performance of various load balancing algorithms in terms of other performance metrics.

## 5.2 Comparison Between Different Algorithms

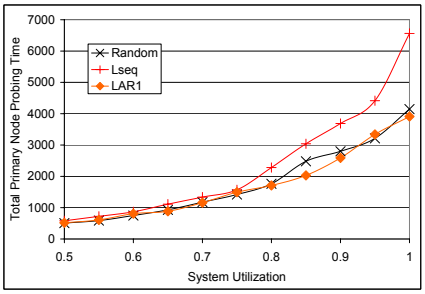
Figure 3(a) shows the probing time of Lseq is much more than $R_1$ and $LAR_1$. This result implies that random algorithm is better than sequential algorithm in probing efficiency. Figure 3(b) shows that the rearrangement number of those three methods are almost the same. This implies that these three algorithms need almost the same number of primary nodes for load rearrangement to achieve load balance. However, more probing time of Lseq suggests that Lseq is not as efficient as random probing. It is consistent with the observation of Mitzenmacher in [5] that simple randomized load balancing schemes can balance load effectively although it is often difficult to analyze such schemes.

Figure 3(c) and (d) show the performance of the algorithms in "ts5k-large". From Figure 3(c), we can observe that bandwidth for probings of $R_1$, $LAR_1$ and Lseq are almost the same in lightly loaded system with utilization no more than 0.6. When system utilization is greater than 0.6, this bandwidth of $R_1$ is more than $LAR_1$ and Lseq, and the performance gap increases as the system load increases. It is because that much less number of probings is needed in lightly loaded system compared with that in heavily loaded system. Such that, probing distance has less effect in bandwidth consumption in lightly load system. The bandwidth results of LAR and Lseq are almost the same when system utilization is no more than 0.9, and when system utilization is more than 0.9, the bandwidth of LAR is more than Lseq's. This is due to the fact that in more heavily loaded system, more randomly chosen nodes from entire Id space need to be probed, which have longer distances to probing nodes than sequential nodes, resulting in more probing bandwidth consumption. Figure 3(d) shows the moved load distribution in global load balancing with system utilization approaches to 1. We can see that $LAR_1$ and Lseq are able to transfer about 60% of total moved load within 10 hops, while $R_1$ transfers only about 15% within 10 hops. That's because $R_1$ is locality-oblivious while $LAR_1$ and Lseq are locality-aware.

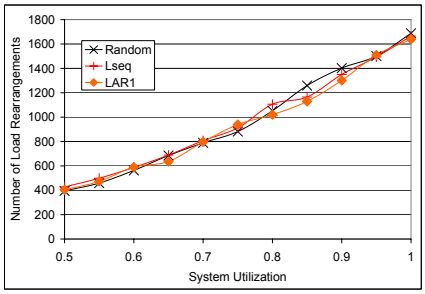
Figure 3(e) and (f) show the performance of algorithms in "ts5k-small". These results also confirm that $LAR_1$ and Lseq achieve better locality-aware performance than $R_1$, although the improvement is not so significant as that in "ts5k-large". It is because that in "ts5k-small" topology, nodes are scattered in the entire network. In this case, the neighbors of a primary node may not be physically closer than other nodes.

In summary, these results suggest that the randomized algorithm is more efficient than the sequential algorithm in the probing process. The locality-aware approaches can effectively assign and transfer loads between neighboring nodes first, thereby reduce network traffic and improve load balancing efficiency. The LAR algorithm performs best among the three algorithms.

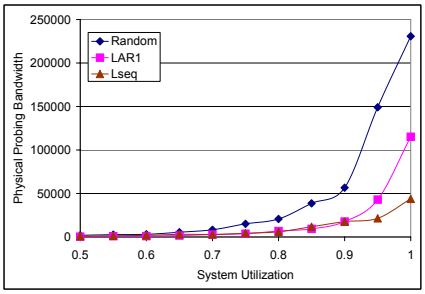
Figures 4 and 5 show the breakdown of total probed nodes in percentage of the probed nodes got from neighbors or from randomly choosing in entire Id space in $LAR_1$ and $LAR_2$ respectively. Label "one neighbor and one random" in Figure 5 represents the condition when there's only one neighbor in routing table, then another probed node is chosen randomly from Id space. We can see that the percentage of neighbor primary node constitutes the most part. With system utilization increases, the percentage of neighbor primary node decreases because the neighbors' DSLs don't have enough free capacity for a larger number of excess items. Therefore, neighbors can support most of system excess items in load balancing, and randomly chosen primary nodes must be resorted to for excess items that cannot be supported by neighbors.



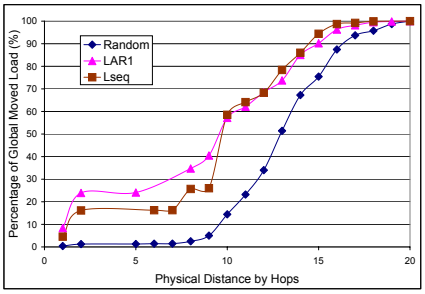

(a) Total primary node probing time "ts5k-large"



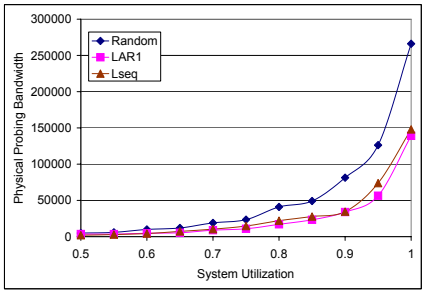

(b) Total number of load rearrangements "ts5k-large"



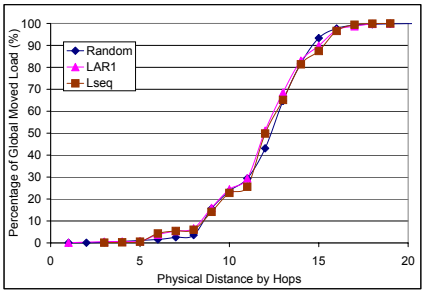

(c) Total bandwidth of probings in "ts5k-large"




(d) CDF of moved load distribution in "ts5k-large"




(e) Total bandwidth of probings in "ts5k-small"




(f) CDF of moved load distribution in "ts5k-small"

**Figure 3. Effect of load balancing due to different probing algorithms**

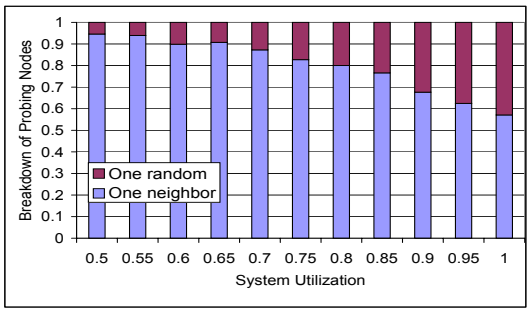



**Figure 4. Breakdown of probed nodes of $LAR_1$**

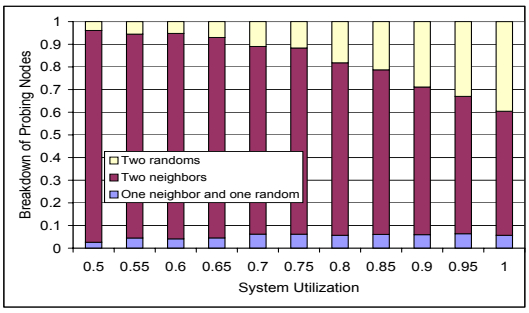



**Figure 5. Breakdown of probed nodes of $LAR_2$**

### 5.3 Effect of D-Way Random Probing

We tested the performance of the $LAR_d$ algorithms with different concurrency degree $d$. Figure 6(a) shows that $LAR_2$ has much less probing time than $LAR_1$. It implies that $LAR_2$ reduces the probing time of $LAR_1$ at the cost of more number of probings. Unlike $LAR_1$, in $LAR_2$, a probing node only sends its SSL to a node with more total free capacity in its DSL between two probed nodes. The more item transfers in one load rearrangement, the less probing time. It leads to less number of SSL sending operation of $LAR_2$ than $LAR_1$, resulting in less number of load rearrangements as shown in Figure 6(b). Therefore, simultaneous probings to get a node with more total free capacity in its DSL can save load balancing time and reduce network traffic load.

From Figure 6(a) and (b), we can observe that the probing efficiency of $LAR_d$ (d>2) is almost the same as $LAR_2$, though they need to probe more nodes than $LAR_2$. Our results are almost in consistent with the observations of randomized algorithms in parallel computing that a two-way probing method leads to an exponential improvement over one-way probing, but a d-way (d>2) probing leads to much less substantial additional improvement [5]. In the following, we will show whether the improvement of $LAR_d$ $(d \geq 2)$ over $LAR_1$ is at the cost of more band-

width consumption or locality-aware performance degradation. Figure 6(c) and (d) show experiment results in "ts5k-large". We can see from Figure 6(c) that the probing bandwidth of $LAR_2$ is almost the same as $LAR_1$. Figure 6(d) shows the moved load distribution in global load balancing of each algorithm. We can see that the distribution of $LAR_2$ is proximately the same as $LAR_1$, and their performance is better than that of $LAR_4$ and $LAR_6$, which are almost the same. This is caused by the fact that the more simultaneous probed nodes, the less possibility that the best primary node is a close neighbor node. Based on these results, we can conclude that $LAR_2$ improves on $LAR_1$ at no cost of bandwidth consumption. It retains the advantage of locality-aware probing.

Figure 6(e) and (f) show the performance of each algorithm "ts5k-small". The nodes in "ts5k-small" are scattered all over the network. This feature leads to less significant result in Figure 6(f). However, we can get the same conclusions as those in "ts5k-large".

### 5.4 Load Balancing in Systems with Churn

In practice, nodes and items continually join and leave P2P systems. It is hard to achieve load balance with churn because of two facts. First, before a node leaves, it transfers all its items to its neighbor, which becomes overloaded if it cannot provide sufficient capacity for those items. Second, continuous and fast file joins increase the probability of overloaded nodes generation. These require an load balancing algorithm to find nodes with sufficient free capacity for excess items quickly in order to keep load balance condition in churn.

We evaluated the efficiency of the LAR algorithms in dynamic situations with respect to a number of performance factors. Experiment results verified the superiority of the algorithm in DHTs with churn, in comparison with a churn resilient algorithm (CRA) proposed in [2]. Due to space constrains, we present a summary of experimental results in this section; more details can be found in [9]. In this experiment, We run each trial of the simulation for 20T simulated seconds, where T is a parameterized load balancing period, and it was set to 60 seconds in our test. The item join/departure rate was modelled by a Poisson process with a rate of 0.4; that is, there were one item join and one item departure every 2.5 seconds. The system utilization of system was set to 0.8. We adopted the same metrics as in [2]:

(1) *99.9th percentile node utilization (99.9th NU).* We measure the maximum 99.9th percentile of the utilizations of the nodes after each load balancing period T in simulation and take the average of those results over a 20T period as the 99.9th NU.
(2) *Load movement factor*, defined as the load moved movement factor corresponding to the 99.9th NU.
(3) *Load moved/DHT load moved (L/DHT-L)*, defined as the total load moved incurred due to load balancing divided by the total load of items moved due to node joins and departures in the system.

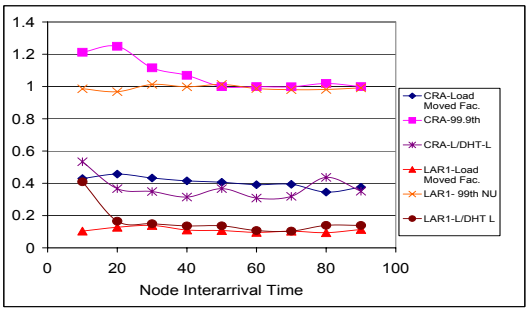


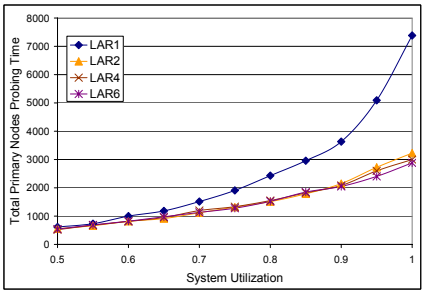

**Figure 7. Effect of load balancing with churn**

Figure 7 plots the performance due to $LAR_1$ and CRA versus node interarrival time. By comparing results of $LAR_1$ and CRA, we can have a number of observations. First, the 99.9th NUs of $LAR_1$ and CRA are kept no more than 1 and 1.25 respectively. This implies that on average, $LAR_1$ can achieve the load balancing goal in churn. Second, $LAR_1$ moves up to $20\%$ and CRA moves up to $45\%$ of the system load to achieve load balance for system utilization as high as $80\%$. Third, the load moved due to load balancing is very small compared with the load moved due to node joins and departures and it is up to 0.4 for $LAR_1$ and 0.53 for CRA. When the node interarrival time is 10, the L/DHT-L is the highest. It is because faster node joins and departures generate much higher degree of load imbalance, such that more load transferred is needed to achieve load balance. The fact that the results of $LAR_1$ are comparable to CRA implies that LAR algorithm is as efficient as CRA to handle churn. In summary, in the face of rapid arrivals and departures of items of widely varying load and nodes of widely varying capacity, LAR algorithm achieves load balance while moving up to $20\%$ of the load that arrives into the system, and up to $40\%$ of the load the underlying DHT moves due to node arrivals and departures. It ensures a load balance condition even in churn.

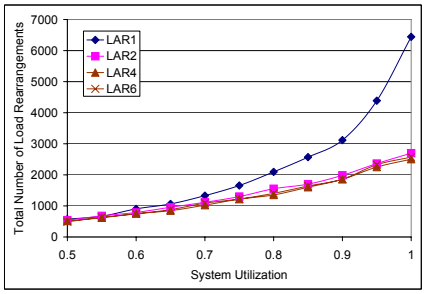
## 6 Conclusions

This paper presents LAR load balancing algorithms to deal with both of the proximity and dynamic features of DHTs. The algorithms distribute application load among the nodes by "moving items" according to node capacities, as well as node proximity information in topology-aware DHTs. We introduce a factor of randomness in the probing process in a range of proximity to deal with DHT churn. We further improve the randomized load balancing efficiency by d-way probing. Simulation results show the superiority of a locality-aware 2-way randomized load balancing in DHTs with and without churn.

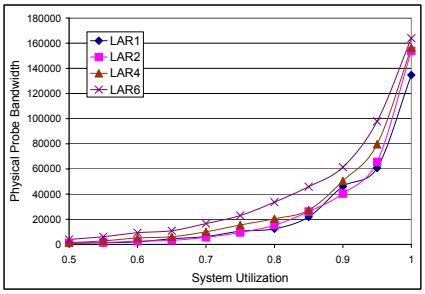
We note that the load balancing algorithms work for key distribution load balancing. In file sharing P2P systems,



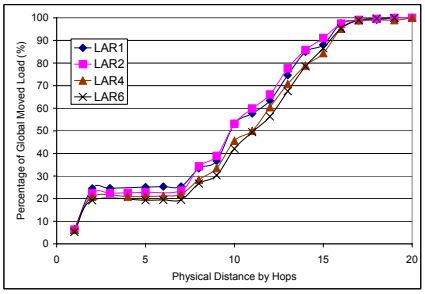

(a) Total primary node probing time "ts5k-large"



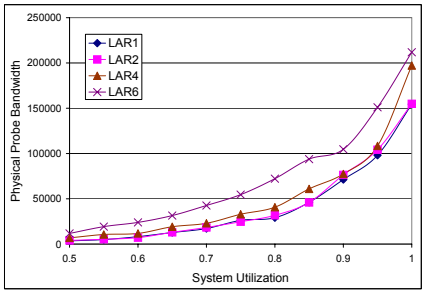

(b) Total number of load rearrangements "ts5k-large"



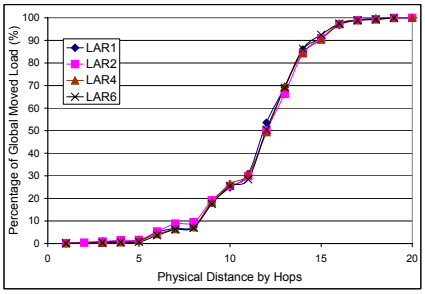

(c) Total bandwidth of probings in "ts5k-large"




(d) CDF of moved load distribution in "ts5k-large"




(e) Total bandwidth of probings in "ts5k-small"




(f) CDF of moved load distribution in "ts5k-small"

**Figure 6. Effect of load balancing due to different LAR algorithms**

a main function of nodes is to handle key location query. Query load balancing is a critical part of P2P load balancing; that is, the number of queries that nodes receive, handle and forward is based on their different capacities accordingly. We will explore methods for this.

### Acknowledgments

This research was supported in part by U.S. NSF grant ACI-0203592 and NASA grant 03-OBPR-01-0049.

## References

[1] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. In *Proc. of ACM STOC*, pages 593–602, 1994.

[2] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *Proc. of IEEE INFOCOM*, 2004.

[3] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and P. R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of STOC*, pages 654–663, 1997.

[4] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *Proc. of IPTPS*, 2004.

[5] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proc. of the 9th ACM SPAA*, pages 292–301, 1997.

[6] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Proc. of IPTPS*, 2003.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 329–350, 2001.

[8] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware*, 2001.

[9] H. Shen and C. Xu. Locality-aware randomized load balancing algorithms for structured p2p systems. Technical report, ECE Department, Wayne State University, 2004.

[10] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree p2p overlay network. *Performance Evaluation*, 2005.

[11] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 1(1):17–32.

[12] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *Proc. of MOBISYS*, 2004.

[13] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *Proc. of ICDCS*, 2003.

[14] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of ICDE*, 2003.

[15] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM*, 1996.

[16] Y. Zhu and Y. Hu. Towards efficient load balancing in structured p2p systems. In *Proc. of IPDPS*, 2004.