Elastic Routing Table with Provable Performance for Congestion Control in 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

Distributed hash table (DHT) networks based on consistent hashing functions have an inherent load balancing problem. The problem becomes more severe due to the heterogeneity of network nodes and the non-uniform and time-varying file popularity. Existing DHT load balancing algorithms are mainly focused on the issues caused by node heterogeneity. To deal with skewed lookups, this paper presents an elastic routing table (ERT) mechanism for query load balancing, based on the observation that high degree nodes tend to experience more traffic load. The mechanism allows each node to have a routing table of variable size corresponding to its capacity. The indegree and outdegree of the routing table can also be adjusted dynamically in response to the change of file popularity and network churn. Theoretical analysis proves the routing table degree is bounded. The ERT mechanism facilitates locality-aware randomized query forwarding to further improve lookup efficiency. By relating query forwarding to a supermarket customer service model, we prove a 2-way randomized query forwarding policy leads to an exponential improvement in query processing time over random walking. Simulation results demonstrate the effectiveness of the ERT mechanism and its related query forwarding policy for congestion and query load balancing. In comparison with the existing "virtual-server"-based load balancing algorithm and other routing table control approaches, the ERT-based congestion control protocol yields significant improvements in query lookup efficiency.

1 Introduction

Because of DHT networks' lookup efficiency, robustness, scalability and deterministic data location, they have received much attention in recent years. DHT networks have an inherent load balancing problem. It is because consistent hashing for file distribution produces a bound of O(log n) key imbalance between the network nodes. The problem becomes even more severe as the nodal heterogeneity increases. What is more, files stored in the system often have different popularities and the access patterns to the same file may vary with time. It is a challenge to design a DHT protocol with the capability of congestion control.

The primary objective of congestion control is to avoid bottleneck in any node (*i.e.* the query load exceeds its capacity). Bottleneck may occur with query overflow in which too many queries received by the node at a time, or with data overflow in which a too high volume of data need to be downloaded and forwarded by the node simultaneously. Although files are often transmitted via a direct connection between source and destination, data forwarding through intermediary nodes in the query routing path is often used for the provisioning of anonymity of file sharing, as in Freenet [6] and Hordes [9].

In the past, many load balancing strategies have been proposed to deal with network heterogeneity; see [8, 3] for examples. They assign a key ID space interval to each node based on its capacity. Because the approaches are based merely on key ID assignment, they do not provide any control over congestions caused by the factor of non-uniform and time-varying file popularity. There are other approaches, based on "item-movement", which take into account the effect of file popularity on query load [2, 15]. In these approaches, heavily loaded nodes probe light nodes and reassign excess load between the peers by changing the IDs of related files or nodes. Albeit flexible, the load reassignment process incurs high overhead for changing IDs, especially under churn.

Notice that the existing load balancing approaches assume that each node (or virtual node) should maintain the same number of neighboring relationships, irrespective of its capacity. The principle of power-law networks tells that higher degree nodes tend to experience more query loads [1]. In light of this, in this paper, we present an elastic routing table (ERT) mechanism to cope with node heterogeneity, skewed queries, and churn in DHT networks. Unlike current structured P2P routing tables with a fixed number of outlinks, each ERT has a different number of inlinks/outlinks and the indegree/outdegree of each node can be adjusted dynamically according to its experienced traffic so as to direct query flow to light nodes. Most recently, Castro *et al.* [4] exploited heterogeneity by modifying the proximity neighbor selection algorithm. The ERT-based congestion control protocol goes beyond the construction of capacity-aware DHTs. It deals with congestion due to time-varying file popularity by adjusting the indegrees and outdegrees of the routing tables and capacity-aware query forwarding. We summarize the contributions of this paper as follows.

- An initial indegree assignment for capacity-aware DHTs construction. The indegrees are provably bounded.

- A policy for periodic indegree adaptation to deal with the non-uniform and time-varying file popularity. It is proved that the indegree bounds remain bounded.
- A topology-aware randomized query forwarding policy on the elastic DHTs. It is proved that the ERT-enabled query forwarding leads to an exponential improvement in query processing time over random walking.
- Comprehensive simulations demonstrate the superiority of the elastic congestion control protocol, in comparison with the "virtual-server"-based load balancing policy and other routing table control approaches.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative congestion control approaches for unstructured and structured P2P systems. Section 3 shows the ERT-based protocol, focusing on initial indegree assignment and periodic adjustment. Section 4 gives the details of topology-aware randomized query forwarding policy. Section 5 shows the performance of the protocol with comparison of a variety of metrics. Section 6 concludes this paper with remarks on possible future work.

2 Related Work

There have been many load balancing algorithms to deal with node heterogeneity and network churn [17, 8, 3]. "Virtual server" [17] is a popular approach, in which each real node runs $O(\log n)$ virtual servers and the keys are mapped onto virtual servers so that each real node is responsible for the key ID space of different length proportional to its capacity. It is simple in concept, but the virtual server abstraction incurs large maintenance overhead and compromises lookup efficiency. Godfrey *et al.*[8] addressed the problem by arranging a real server for virtual ID space of consecutive IDs. In [3], Bienkowski *et al.* proposed a distributed randomized scheme to let a linear number of nodes with short ID space interval to divide the existing long ID space interval, resulting in an optimal balance with high probability.

Initial key ID space partitioning is insufficient to guarantee load balance, especially in churn. It is often complemented by dynamic load reassignment. Godfrey *et al.* proposed schemes to rearrange load between heavy nodes and light ones so as to avoid bottleneck [7]. They assumed that query load was uniformly distributed in the ID space. Bharambe *et al.* [2] proposed a load balancing algorithm to deal with the congestion caused by biased lookups. It proceeds in a way that heavily loaded nodes requests lightly loaded nodes to leave from their current locations and rejoin at the location of the heavily loaded nodes.

Castro *et al.* [4] proposed a neighbor selection algorithm to construct routing tables based on node capacities. Its basic idea of using node indegrees to exploit node heterogeneity is similar to our initial indegree assignment algorithm. Their algorithm directs most traffic to high capacity nodes because it does not choose low capacity nodes as neighbors unless the indegree bounds of high capacity nodes are reached. In contrast, ERT mechanism would distribute the traffic between the neighbors proportional to their capacities so as to make full utilization of both high and low capacity nodes. More importantly, ERT mechanism deals with more than node heterogeneity. It deals with biased lookups and network churn by adjusting the table indegree and outdegree dynamically and query forwarding.

Finally, we note that the problem of congestion control is not unique in structured P2P networks. It has been a crucial performance issue in unstructured P2P networks, as well. Many studies have been devoted to flow control in unstructured networks; see [10, 5] for recent examples. Like congestion control in DHT networks, their solutions are based on the principle of power-law networks as well. Since they were designed for flooding or random walk query routing networks, the flow control algorithms on unstructured P2P networks cannot be applied for load balancing and congestion control in DHT networks.

3 Elastic DHT

ERT is designed based on the principle of power-law networks that a higher degree node tends to receive more query load. The ERT-based protocol constructs routing tables of different number of outlinks so as to distribute query load among the nodes in proportion to their capacities. Each node dynamically adjusts its indegree according to its actual query load.

3.1 Query Load Balancing and ERT

We assume a DHT network with n physical nodes, labelled as an integer from 1 to n. Node i, $1 \leq i \leq n$, has a capacity that it is willing to devote or able to process queries. We assume that node i's capacity c_i is a quantity that represents the number of queries that node i can handle in a given time interval T. In practice, the capacity should be determined as a function of a node's access bandwidth, processing power, disk speed, etc. We define the load of node i, l_i, as the number of queries it receives and transmits to its neighbors over time T. We refer to node with traffic load $l_i \leq c_i$ as a *light node*; otherwise a heavy or overloaded node.

The purpose of a congestion control protocol is to avoid heavy nodes in query routings and distribute query load among nodes corresponding to their capacities. From the view point of an entire system, fair load distribution is achieved by letting each node's load share proportional to its "fair load share", as defined by $s_i = \frac{l_i / \sum_i l_i}{c_i / \sum_i c_i}$. Ideally, the fair share s_i should be kept close to 1. One way to achieve this is to measure the traffic load l_i of every node periodically and forward queries according to collected global traffic load information. Obviously, this method is too costly to be used in any scalable overlay networks. In [10], Lv *et al.* showed that a high degree node in Gnutella network would most likely experience high query load. We apply the same principle to the design of congestion control in DHT networks. We define *indegree*, denoted by d_i, $1 \leq i \leq n$, as the number of inlinks of node i. Under the assumption that nodes and file queries are uniformly distributed in a DHT network without churn, l_i is directly related to d_i. For that reason, we propose to set the initial indegree of each node to an appropriate value, according to its capacity. As a result, heterogenous nodes would have routing tables of different sizes (outdegree). It is expected that the higher the indegree of a node, the higher traffic load the node would experience in the uniform system. To deal with skewed queries caused by non-uniform and time-varying file popularity and network churn,

we propose an integral indegree adaptation component to dynamically adjust node indegree periodically. We refer to such an elastic routing table as ERT.

3.2 Initial Indegree Assignment

In current DHT networks, each routing table has a fixed number of outlinks for a given network size n. For example, it is $\log n$ in Chord and a constant number in Cycloid [16]. The outlinks determine node indegrees. Since we want to have a node's indegree to be proportion to its capacity, we reverse the relationship to determine node outdegree by setting an appropriate value of indegree. To the end, we need to address two questions:

1. How to determine a node's indegree to make full use of its capacity and keep it lightly loaded meanwhile?
2. How to construct ERTs with different indegrees for nodes of different capacities, and meanwhile retain the original DHT neighbor selection functions in lookup ?

We normalize capacity to c so that the average of c is 1; that is, $\sum_i c_i = n$. Recall that in a uniform system, l_i is related to d_i directly. It follows that $s_i \approx \frac{d_i/\sum_i d_i}{c_i/\sum_i c_i}$, and $d_i \approx c_i \frac{\sum_i d_i}{n}$. Taking $\sum_i d_i/n$ as a constant α, we define α as *indegree per unit capacity*. It is a system parameter and is determined as a function of different metrics in system experience such as inlink query forwarding rate, query initiation rate, etc. in non-uniform systems. We first set node i's maximum indegree d_i^∞ as $\lfloor 0.5 + \alpha c_i \rfloor$ based on the fact that the maximum number of queries a node can process at a time depends on its capacity. The initial indegree of node i is βd_i^∞, where β is a pre-defined percentage for reservation purpose.

In the following, we will explain how to build ERTs with initial setting of the indegree for each node. Like bi-directional links in Gnutela, we let each DHT node i maintain a backward outlink (*backward finger*) for each of its inlink. Once node i joins the system, it builds its routing table based on DHT protocols. In order to control d_i below d_i^∞, we set a restriction that only nodes with available capacity $d_i^\infty - d_i \geq 1$ can be the joining node's neighbors. After building a basic routing table, node i then probes other nodes who can take it as their neighbor to achieve its initial indegree βd_i^∞. As a result, high capacity nodes produce high indegrees while low capacity nodes lead to low indegrees.

To probe nodes for indegree expansion, the IDs of those nodes should be firstly decided. The ID set can be determined in the opposite way of the original DHT neighbor selection algorithm. For example, in Cycloid with dimension d, a node $(k, a_{d-1}a_{d-2}\ldots a_k\ \ldots a_0)$ $(k \neq 0)$ has one cubical neighbor $(k-1, a_{d-1}a_{d-2}\ldots \overline{a}_k xx...x)$, and two cyclic neighbors $(k-1, b_{d-1}b_{d-2}\ldots b_0)$ and $(k-1, c_{d-1}c_{d-2}\ldots c_0)$ in its routing table. By the opposite way of neighbor selection, a node $(k-1, a_{d-1}a_{d-2}\ldots a_k\ldots a_0)$ can send requests targeting to $(k+1, a_{d-1}a_{d-2}\ldots \overline{a}_k xx...x)$ to ask nodes to take it as their cubical neighbors, and also it can send requests targeting to $(k+1, a_{d-1}a_{d-2}\ldots a_k xx...x)$ to ask nodes to take it as their cyclic neighbors. Algorithm 1 shows the pseudocode of indegree expansion algorithm in Cycloid. In the pseudocode, we represent represent $a_{d-1}a_{d-2}\ldots a_k\ldots a_0$ in node ID $(k, a_{d-1}a_{d-2}\ldots a_k\ldots a_0)$ as a_{id}. On receiving a request, the node which is responsible for the ID checks if it can take node i as its routing table neighbor. If so, it adds i into its corresponding entry in its routing table and sends back a positive reply. Once node i receives positive reply from a node, say node j, it builds a backward finger to node j. The algorithm can be applied to Chord, Pastry, Tapestry and other structured overlays that have little flexibility in the selection of neighbors by relaxing their routing table neighbor constraints. Readers are referred to [14] for the details.

Algorithm 1 Pseudo-code for indegree expansion algorithm of Cycloid node i $(k, a_{d-1}a_{d-2}\ldots a_k\ldots a_0)$.

```
1: //probe backward fingers of cubical neighbor
2: figure out a set of cubical neighbor inlinks ID = (k+1, a_{d-1}a_{d-2}...ā_k xx...x)
3: id = (k+1, a_{d-1}a_{d-2}...ā_k 00...0)
4: while not finish probing all IDs in ID ∧((d_i^∞ − d_i) ≥ βd_i^∞) do
5:    while id is in backward fingers do
6:       id = (k+1, a_id++) ∈ ID
7:    end while
8:    probe id for cubical neighbor inlink
9:    id = (k+1, a_id++) ∈ ID
10: end while
11: //probe backward fingers of cyclic neighbor
12: figure out ID of cyclic neighbor inlinks ID = (k+1, a_{d-1}a_{d-2}...a_k xx...x)
13: id = (k+1, a_{d-1}a_{d-2}...a_k 00...0)
14: while not finish probing all IDs in ID ∧((d_i^∞ − d_i) ≥ βd_i^∞) do
15:    while id is in backward fingers do
16:       id = (k+1, a_id++) ∈ ID
17:    end while
18:    probe id for cyclic neighbor inlink
19:    id = (k+1, a_id++) ∈ ID
20: end while
```

The initial indegree assignment algorithm proceeds recursively. We prove the ERT indegree resulted from the algorithm is bounded. Readers are referred to [14] for the proof of all theorems. We assume a DHT that manages a unit-size ID space $[0, 1)$, and the DHT uses consisting hash to partition the ID space among the nodes. Thus, the responsible ID space imbalance is $\log n$. We assume that each node i can estimate its capacity c_i and the network scale n within a factor of γ_c and γ_n, respectively, of the true values, with high probability[1]; readers are referred to [11] for details of such an estimation process.

Theorem 3.1 *The initial indegree assigned to a node i is between $\alpha c_i/\gamma_c - O(1)$ and $\alpha c_i \gamma_c + O(1)$ w.h.p.*

3.3 Periodic Indegree Adaptation

Considering the fact that query load often varies with time, the initial indegree assignment is not robust enough to limit a node's query load under its capacity. The congestion control protocol should adapt to the change of query rate and lookup skewness caused by non-uniform and time-varying file popularity, as well as network churn.

[1]An event happens with high probability (w.h.p.) when it occurs with probability $1 - O(n^{-1})$.

We design an periodic indegree adaptation algorithm to help each node adjust its indegree periodically according to the maximum load it experienced. Specifically, every node i records its query load l_i over T periodically and checks whether it is overloaded or lightly loaded by a factor of γ_l; *i.e.* whether $g_i = l_i/c_i > \gamma_l$ or $< 1/\gamma_l$. In the former case, it decreases $\mu(l_i - c_i)$ indegree by asking some of its backward fingers to delete it from their routing table, then deletes corresponding backward fingers, and decreases its maximum indegree d_i^{∞} correspondingly. To choose backward figures for removing, it chooses the backward figure with longest logical distance. In the latter case, it increases $\mu(c_i - l_i)$ indegree by probing other nodes to take it as their neighbors using the inlink expansion algorithm discussed in Section 3.2 and increases its d_i^{∞} correspondingly.

The following theorems show that the ERT indegree and outdegree in the process of adaption remain bounded.

Theorem 3.2 *With indegree assignment and periodic adaptation algorithm, a node i has an indegree between $\frac{c_i}{\gamma_c\gamma_l\nu_{max}}$ and $\frac{c_i\gamma_c\gamma_l}{\nu_{min}}$ where ν_{max} and ν_{min} represent the maximum and minimum incoming query rate per inlink in the system, respectively. And its indegree change is bounded in each adaptation.*

Theorem 3.3 *A Cycloid node has an outdegree of at most $\frac{2\gamma_c\gamma_l c_{max}}{\nu_{min}} - O(\frac{2^d}{d}) + O(1)$ w.h.p. where d is the DHT dimension.*

4 Topology-aware Randomized Query Forwarding

Periodic indegree adaptation is not sufficient to deal with query load imbalance caused by churn and skewed lookups. In this section, we present a complementary topology-aware randomized query forwarding algorithm to help forward queries towards light nodes, and meanwhile reducing lookup latency.

4.1 Query Forwarding

With the initial indegree assignment and periodic adaptation algorithms, each node's routing table has a variable size. With a high probability, each ERT has a set of outlinks in each of its routing table entries. For example, a Cycloid node i (4,101-1-1010) has cubical outlinks pointing to nodes (3,1010-0000), (3,1010-0001) and (3,1010-0010). If it receives a query and decides that the query be forwarded to its cubical neighbors based on its original routing algorithm, there would be three candidates to take the query.

A simple forwarding policy is random walk, in which one of the outlinks is selected randomly. Another one is gradient-based walk that forwards the query to the "best" candidate in terms of their workload. Instead of probing all of the neighbors to find out the best candidate, we restrict the search space to a small set of size b. That is, once receiving a query, node i first randomly selects b neighbors (outlinks) and then probes the nodes in the set sequentially, until a light node is found. In the case that all candidates are overloaded, the query is forwarded to the least heavily loaded one.

The b-way randomized query forwarding is further enhanced by taking into account the underlying topology information in the candidate selection. In the topology-aware forwarding policy, a node selects the best candidate among b neighbors by two extra criteria: close to the target ID by the logical distance (hops) in the DHT network and close to the node by the physical distance on the Internet; Readers are referred to [15] for a landmarking method to measuring physical distance between two nodes on DHT networks. In the case that the two candidates are both lightly loaded, the closer node in logical distance is selected. Their physical distance is used to break the tie of logical distance.

Probing b neighbors is a costly process. How to find a good candidate from b neighbors at a relatively low cost? Query forwarding in this context can be regarded as a supermarket customer service model. The supermarket model is to allocate each incoming task (a customer) to a lightly loaded server with the objective of minimizing the time each customer spends in the system. Mitzenmacher [12] proved that granting a task with two server choices and dispatching it to one of the servers with less workload lead to an exponential improvement over the single choice in the expected execution time of each task. But a poll size larger than two gains much less substantial extra improvement. Furthermore, Mitzenmacher *et al.* [13] improved the performance of two-choice method dramatically by the use of memory. In this method, each time a task is allocated, the least loaded of that task's choices after allocation is remembered and used as one of the possible choices for the next task. We tailored this memory-based randomized task dispatching method with modifications to topology-aware randomized query forwarding. We set $b = 2$. A node first randomly selects two options, say node i and j. It then selects the better one, say node i, and the least loaded node between i and j is remembered after node i increases by one load unit. We assume the node is still i, which is used for the next forwarding. Later, when the node needs to forward a query to the same routing table entry, it only needs to randomly choose one neighbor, instead of two. With the remembered i, it starts the process again.

To further reduce the heavy nodes in query routing, a query flows by the use of the information of overloaded nodes encountered before, to avoid overloaded node in the succeeding routing. Algorithm 2 shows the pseudocode of the topology-aware randomized query forwarding algorithm.

4.2 Analysis of Query Forwarding

The simple query forwarding model (QFM) can be rephrased as the following: after a node receives a query, it forwards the query to one of its neighbor options. If the chosen neighbor is heavily loaded by a factor γ_l, another specific neighbor is turned to. This process is repeated until the node finds a light neighbor. In the case that all neighbor options are heavy, the query is forwarded to the least heavily loaded option. We assume that the query forwarding time for a query is constant and incoming query is Poison distributed.

The forwarding model can be regarded as a variation of *strong threshold supermarket model* (STSM) proposed in [12] if we take γ_l as the threshold T in the latter. In the STSM, customers arrive at a Poison stream of rate λn ($\lambda < 1$) at n FIFO servers. Each customer chooses a server independently and uniformly at random and only makes additional choices if the previous choice is beyond a pre-determined threshold. If both choices are over the threshold, the cus-

Algorithm 2 Pseudo-code for topology-aware randomized query forwarding algorithm executed by node i.

```
receive query Q with overloaded node information A
determine the set of outlinks for the query forwarding based on DHT routing algorithm.
choose options J = {j_1, j_2 ...} from the outlink set excluding overloaded node in A
if memory has a node j_a then
    randomly choose a node j_b from J
else
    randomly choose two nodes j_a and j_b from J
end if
//choose the better node from j_a and j_b
probe node j_a and j_b for load status
if j_a and j_b are heavy then
    add j_a and j_b to A
    forward Q and A to the least heavily loaded node
else
    if one node is light and one node is heavy in j_a and j_b then
        add the heavy node to A
        forward Q and A to the light node
    end if
else
    choose nodes J_log logically nearest to target ID from j_a and j_b
    choose nodes J_phy physically nearest to node i from J_log
    forward Q and A to a node in J_phy
end if
```

tomer queues at the shorter of its two choices. The service time for a customer is exponentially distributed with mean 1. A key difference between the QFM and the STSM is that the servers are homogeneous in the supermarket model but heterogeneous in the query forwarding model.

Along the line of analytical approach in [12], we analyze the performance of the query forwarding algorithm. The following theorem shows that the 2-way randomized query forwarding improves lookup efficiency exponentially over random walking.

Theorem 4.1 *For any fixed time spot T, the time a query waits before being forwarded during the time interval $[0, T]$ is bounded and b-way $(b \geq 2)$ forwarding yields an exponential improvement in the expected time for a query queuing in a server.*

5 Performance Evaluation

This section demonstrates the distinguished peoperties of the ERT-based congestion control protocol through simulation built on an $O(1)$-degree Cycloid network. Simulations on other $O(\log n)$-degree networks are expected to produce better results. Table 1 lists the parameters of the simulation and their default values. The bounded Pareto distribution for the capacity of nodes reflects real world situations where machines' capacities vary by different orders of magnitude. The queries are consecutively generated with a random source node and a random target key, unless otherwise noted. We evaluate the effectiveness of the congestion control protocol in the following metrics:

- Congestion rate of a node i, as defined by $g_i = l_i/c_i$. Ideally, the congestion rate should be kept around 1, implying the node is neither overloaded nor under utilized. We use the metric of *the 99th percentile maximum congestion* to measure the network congestion, and use the metric of *the 99th percentile congestion of minimum capacity node* to reflect the node utilization.
- Query distribution share s_i. It represents the performance of fair load distribution. The objective of fair sharing is hard to achieve in DHT networks because of the DHT strictly controlled topology, routing algorithm, non-uniform and variable file popularity, and churn. We use the metric of *the 99th percentile share* to show how it can be approximated by the control of indegrees.
- Query processing time. It is determined by two factors: lookup path length and the number of heavy nodes encountered in each path. The metric of path length reflects the performance of the query forwarding algorithm, and the metric of number of heavy nodes shows how the congestion control protocol avoids heavy nodes in direct traffic flow.

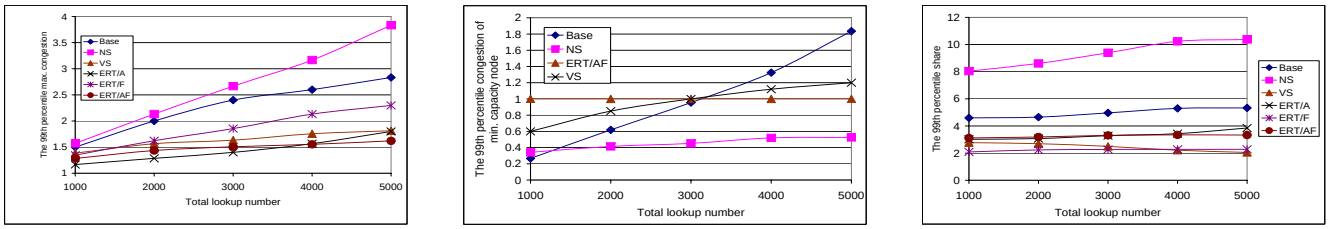
Table 1. Simulation parameters.

Environment Parameter	Default value
Cycloid dimension d	8
Number of nodes n	Fixed at 2048
Node capacity c	Bounded Pareto: shape 2 lower bound: 500 upper bound: 50000
Query/lookup number	3000
Overload threshold γ_l	1
Indegree adaption constant μ	$1/2$
Indegree adaption period	1 second
Indegree per normalized capacity α	dimension d+3
Query process time in light nodes	0.2 second
Query process time in heavy nodes	1 second

We conducted experiments on Cycloid networks without congest control (Base) and with ERT-based congestion control (ERT). For comparison, we include the results due to a "virtual server" load balancing method [8] (VS) and a neighbor selection algorithm for indegree control (NS) [4]. ERT allows dynamic indegree adaption (A) and facilitates query forwarding (F) to deal with network churn and skewed lookups. We represent the congestion control in different combinations by ERT/A, ERT/F, and ERT/AF, respectively. We measured their performance as functions of total lookup number and query processing speed at each node. We varied lookup number from 1000 to 5000, with 1000 increase in each step. We also varied the processing time of a query in a light node from 0.1 second to 2.1 second and 5 times of that in a heavy node. The total query load increases in both cases and we observed similar results in simulation.

5.1 Congestion Control Efficiency

We measured each node's maximum congestion during all test cases and calculated the 99th percentile maximum node congestion. Figure 1(a) shows the congestion rate due to each method increases as more lookup queries arrive. The NS protocol produces a higher rate than Base. It implies that a heavy node in NS has much more load corresponding to its capacity than a heavy node in Base. This is expected because NS strongly biases high capacity nodes as routing table



(a) Maximum congestion (b) Congestion of minimum capacity node (c) Share

Figure 1. Effectiveness of congestion controls.

neighbors, which may turn out to be overloaded. In contrast, VS and ERT/AF demonstrate effectiveness in controlling the load of each node based on its capacity. Although ERT/A and ERA/F do not perform as well as VS, the combined effect of adaptation and forwarding makes ERT/AF outperform VS.

The relative performance between NS, VS, and ERT/AF with respect to Figure 1(a) can be verified by the 99th percentile congestion rate of minimum capacity node in Figure 1(b). Without congestion control (Base), the congestion rate increases sharply. The congestion control protocols delay the occurrence of congestion. In particular, the NS protocol over-protected low capacity nodes due to its high capacity-biased neighbor selection policy. In comparison, ERT/AF keeps low capacity nodes fully utilized, without driving them into overloaded states.

Figure 1(c) shows the 99th percentile node share. We can see that NS generates a much higher share rate, in comparison with the other protocols for the same reason of the observations in Figure 1(a) and (b). In contrast to NS, VS and ERT/AF do not have this preference in neighbor selection and they achieve good query load sharing between heterogeneous nodes. The small gain of VS is due to its fine grained ID space partition between virtual servers. However, it is at the cost of more maintenance overhead and lookup cost. It cannot handle skewed lookups either. The results also confirm the superior performance of ERT/F, which controls query flow to light nodes. ERT/A also helps fair load balancing, though the improvement is not so much as ERT/F.

5.2 Lookup Efficiency

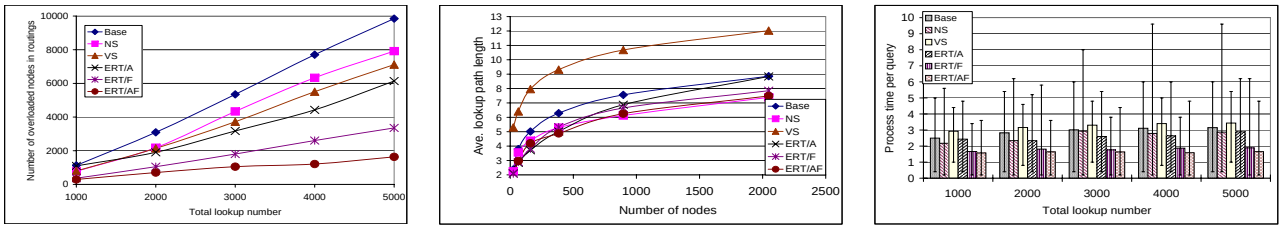
Lookup latency is determined by two factors: lookup path length and query processing time in each node along the path. Figure 2(a) shows that ERT/AF leads to much high lookup efficiency in comparison with the others. Though NS and VS improve over Base to a certain extent, there remain a large percentage of congested nodes in the systems in comparison with ERT/AF. From the figure, we can observe that both ERT/F and ERT/A greatly help eliminate heavy nodes. Their combination demonstrates an accumulated effect. NS biases high capacity nodes for query load, which may make them more likely overloaded as the system query load increases. Due to DHT's strictly controlled topology and precise lookup algorithm, the assumption of uniformly distributed load of VS does not hold true. The fixed outdegree of nodes in NS and VS prevents each node from adapting traffic load. In contrast, ERT/AF enables each node match its indegree to its capacity, and adapts its indegree in response to the change of its experienced query load. Furthermore, the query forwarding operation helps avoid overloaded nodes during query routing.

Figure 2(b) shows the path lengths due to different congestion protocols as the network size increases. It is expected that VS leads to a much longer query path length than Base because of the additional virtual server layer in routing. This is consistent with the observation in [8] that VS achieves the objective of load balancing at the cost of lookup efficiency. ERT/AF takes distance into account in query forwarding. Likewise, NS considers node distance in neighbor selection. Both of them reduce path length of Base significantly. The results also shows that ERT/F leads to a short lookup path, but ERT/A has no effect in path length.

Figure 2(c) shows the average, 1st and 99th percentiles of processing time per query as combined effect of reduced congested nodes and lookup path length on the overall query processing time. Although VS reduces the number of congested nodes of Base, its benefits may be outweighed by its extended path length. Without dynamic congestion control, Base and NS may forward queries to congested nodes. Static indegree assignment by NS only results in marginal processing time reduction. On the contrary, ERT/AF tends to direct queries to light nodes, which have sufficient capacity to handle them promptly.

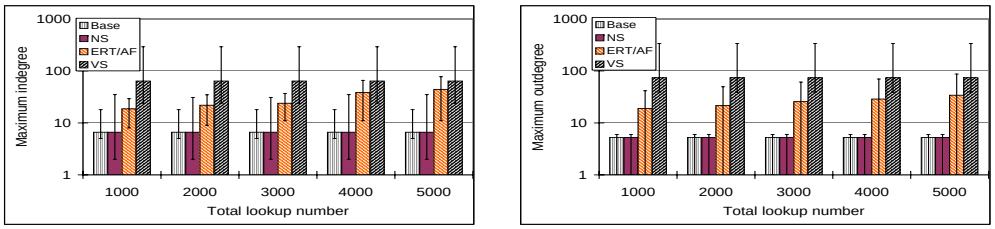
5.3 Maintenance Cost

Recall that ERT-based congestion control protocol achieves its goal using ERTs to adapt each node indegree to its load. In addition to maintain the tables, each node needs to maintain a list of backward fingers. We measured the maximum indegree and outdegree of each node, and calculated the average, 1st and 99th percentiles of those values in each method. We use maximum indegree and outdegree for the evaluation of the management overhead of ERT in the worse case. Figure 3(a) and (b) plot the results. Though inlinks in Base and VS don't need to be maintained, we include their degrees for comparison. As expected, the figures show that the indegree and outdegree rates of Base, NS and VS do not change, while the rates of ERT/AF change as total query load changes. Because ERT tunes node indegees to adapt to different query load accordingly, some light node indegrees reach a high value to request more load. Indegree change leads to outdegree change. The indegree and outdegree of VS are much higher than others. The reason is virtual node usage leads to larger overlay size. Our results turn out that the combination of the average, 1st and 99th percentiles indegree and outdegree of ERT in the worse case is much less than the outdegree rates of VS, respectively. Thus, to achieve congestion control, VS needs much higher cost for maintenance, while



(a) Heavy nodes in routings (b) Lookup path length (c) Lookup time

Figure 2. Effectiveness of congestion control protocols on lookup efficiency.



(a) Indegree (b) Outdegree

Figure 3. Degrees of routing tables in different congestion control protocols.

ERT only needs a little extra maintenance cost.

5.4 Effect of Skewed Lookup

We consider an "impulse" of 100 nodes whose IDs are distributed over a contiguous interval of the ID space, and whose interest are in the same 50 keys randomly chosen from the ID space. Figure 4(a) and (b) plot the overloaded nodes in routings and the query processing time of each method, respectively. It is surprising to see that the overloaded node number and the process time per query of VS is much more than Base. As claimed by the authors in [8] that a good balance of VS is guaranteed only under the uniform load assumption; this explains why VS has poor performance in skewed lookups. In VS, when query load concentrates on a certain ID space interval, the load is allocated to consecutive virtual servers which may reside on the same real node, the node more likely becomes overloaded. In contrast, by assigning and adjusting node indegree based on load dynamically, combined with query forwarding algorithm, ERT/FA can handle skewed lookups caused by the change of file popularity and node interests. NS yields a similar lookup latency to Base on average, but exhibits a large variance.

Figure 4(c) plots the 99th share of each method. By comparing it with Figure 1(b), we can observe that the share rate of each method is higher in skewed lookups. It is expected because the query load concentrates on certain ID space part, then certain nodes. The share rate of NS is still much higher than others in skewed lookups because of its strong bias toward high capacity notes in neighbor selection.

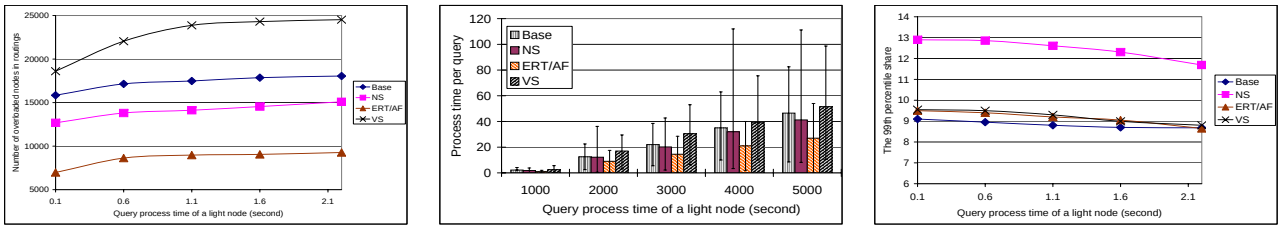
5.5 Effect of Churn

This section evaluates ERT/FA's adaptability to different levels of churn. In this experiment, the lookup rate was modelled by a Poisson process with a rate of 1. The node join/departure rate was also modelled by a Poisson process. We ranged node interarrival/interdeparture time from 0.1 to 0.9 seconds, with 0.1 second increment in each step. Lower time corresponds to higher churn.

Figure 5(a) shows the 99th percentile maximum congestion of each method. The rates of NS and Base grow inversely proportional to node interarrival time, and the rates of VS and ERT/FA maintain constant. When node interarrival time is 0.1, the rate of NS is higher than Base's, and it decreases slightly blow the Base's when node interarrival time is 0.3-0.5. This implies that NS has difficulty to cope with high churn. Recall that in NS, high capacity nodes have denser inlinks, and they may don't have enough capacity for a sudden query flow in high churn. In high churn, VS has marginally less rate than Base, which implies that VS can deal with churn to a certain extend. We can also see that ERT/FA keeps the rate close to 1 in different levels of churn.

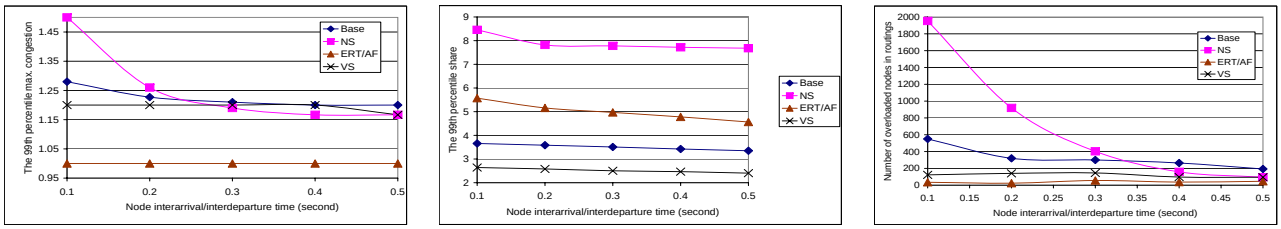
Figure 5(b) shows the 99th percentile share of each method. It demonstrates that like in static DHT, NS performs not so well as others in fair load balance in churn. The share rate of ERT/FA in churn is higher than that without churn. It is because continuous node joins and departures induce more load on some nodes relative to their capacity. On the other hand, because of churn, NS's share rate is higher than Base, and VS has more balanced query load distribution than others.

Figure 5(c) shows the heavy node number in routings of each method in churn. We can see that the number of NS is much higher than Base in high churn, and the number decreases as the node interarrival time increases; both of them are larger than the result of ERT/AF. This observation is consistent to the findings in Figure 5(a). It confirms that ERT/FA performs the best in reducing heavy nodes processing query. Simulation results show the lookup path lengthes of NS and VS in churn are almost the same as those in Figure 2(b) respectively, and ERT/FA has shorter path length in churn. The average, 1st and 99th percentiles of query processing time per node of each method in churn are consistent to those without churn in Figure 2(c), except that NS yields higher latency than Base in high churn. It validates the conclusion that NS is not efficient in coping with high churn.



(a) Heavy nodes in routings (b) Lookup time (c) Share

Figure 4. Effectiveness of congestion control protocols in skewed lookups.



(a) Maximum congestion (b) Share (c) Heavy nodes in routings

Figure 5. Effectiveness of congestion control protocols in networks with churn.

6 Conclusions

DHT networks have an inherent congestion problem caused by query load due to the nature of heterogeneity and dynamism of network nodes. This paper presents a ERT-based congestion control protocol for DHT networks, which consists of three components: indegree assignment, periodic indegree adaptation, and topology-aware query forwarding. Theoretical analysis establishes the bounds of the indegree and outdegree, and proves the performance of the protocol in general in terms of both query load balance factor and query processing time. Simulation results show the superiority of the congestion control protocol compared with other methods in static network, skewed lookups and in churn, and show the effectiveness of each algorithm in the protocol. It makes full use of each node's capacity and meanwhile controls each node' load below its capacity. It improves the lookup efficiency in DHT network by reducing lookup latency.

Acknowledgement

The authors thank anonymous reviewers for constructive comments. This research was supported in part by U.S. NSF grant ACI-0203592, CCF-0611750 and NASA grant 03-OBPR-01-0049.

References

[1] L. A. Adamic, B. A. Huberman, R. M. Lukose, and A. R. Puniyani. Search in power law networks. In *Physical Review E*, volume 64, pages 46135–46143, 2001.

[2] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *Proc. of ACM SIGCOMM*, 2004.

[3] M. Bienkowski, M. Korzeniowski, and F. M. auf der Heide. Dynamic load balancing in distributed hash tables. In *Proc. of IPTPS*, 2005.

[4] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proc. of NSDI*, 2005.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM*, 2003.

[6] I. Clarke and O. Sandberg et al. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. International Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2001.

[7] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. *Performance Evaluation*, 63(3), 2006.

[8] B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *Proc. of INFOCOM*, 2005.

[9] B. Levine and C. Shields. Hordes: A multicast-based protocol for anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[10] Q. Lv, S. Ratnasamy, and S. Shenker. Can heterogeneity make gnutella scalable? In *Proc. of IPTPS*, 2002.

[11] G. Manku. Balanced binary trees for ID management and load balance in distributed hash tables. In *Proc. of PODC*, 2004.

[12] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proc. of SPAA*, 1997.

[13] M. Mitzenmacher, B. Prabhakar, and D. Shah. Load balancing with memory. In *Proc. of FOCS*, 2002.

[14] H. Shen and C. Xu. Elastic routing table for congestion control in chord. Technical report, Electrical and Computer Engineering Dept, Wayne State University, 2005.

[15] H. Shen and C. Xu. Locality-aware randomized load balancing algorithms for structured p2p networks. In *Proc. of ICPP*, pages 529–536, 2005.

[16] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree p2p overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[17] I. Stoica and R. Morris et al. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 2003.