Elastic Routing Table with Provable Performance for Congestion Control in DHT Networks

Haiying Shen, *Member*, *IEEE*, and Cheng-Zhong Xu, *Senior Member*, *IEEE*

Abstract—Consistent hashing-based DHT networks have an inherent load balancing problem. The problem becomes more severe in heterogeneous networks with nonuniform and time-varying popular files. 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 receive more traffic load. The mechanism allows each node to have a routing table of variable size corresponding to node capacities. 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 that 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 that a two-way randomized query forwarding policy should lead 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 existing "virtual-server"-based load balancing algorithms and other routing table control approaches, the ERT-based congestion control protocol yields significant improvement in query lookup efficiency.

Index Terms—Distributed hash table, peer-to-peer, load balancing, congestion control.

1 Introduction

In structured P2P overlay networks, each node and file key is assigned a unique ID, based on a consistent hashing function. The file keys are mapped on to nodes according to their IDs and a distributed hash table (DHT) definition. The DHT maintains topological relationships between the nodes and supports a routing protocol to locate a node responsible for a required key. Because of their lookup efficiency, robustness, scalability, and deterministic data location, DHT networks have received much attention in recent years. Representatives of the systems include CAN [26], Chord [33], Tapestry [34], Pastry [27], and Cycloid [32].

DHT networks have an inherent load balancing problem. It is because consistent hashing produces a bound of O(log n) imbalance degree of keys 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 needed to be downloaded and forwarded by the node. DHT routing algorithms may lead to the convergence of query load targeted for an object on a small number of nodes around the destination, leading to bottlenecks. In addition, 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 [10], Mantis [5], Mutis [1], and Hordes [16]. Recent studies of P2P file sharing systems [13], [28] demonstrate that node capacity and node query pattern are heavily skewed in the systems. Nodes are easily become bottlenecks in such an environment.

In the past, many load balancing strategies have been proposed to deal with the network heterogeneity; see [33], [12], [4] for examples. It is known that a long key ID space interval has a high probability of being contacted than a short interval. Most of the existing approaches share a common idea of "virtual server," in which a physical node simulates a number of virtual overlay servers so that each node is assigned ID space interval of a different length according to its capacity. The simplicity of the approaches comes at a high cost to maintain the relationship between a node's responsible interval and its capacity. Moreover, because the approaches are based merely on key ID assignment, they do not provide any control over congestions caused by the factors of nonuniform 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 [3], [31], [30]. In these approaches, heavily loaded nodes probe light ones and reassign excess load between the peers by changing the IDs of related files or nodes. Albeit

- *H. Shen is with the Department of Electrical and Computer Engineering, Clemson University, 313-B Riggs Hall, Clemson, SC 29634. E-mail: shenh@clemson.edu.*
- *C.-Z. Xu is with the Department of Electrical and Computer Engineering, Wayne State University, 5050 Anthony Wayne Drive, Detroit, MI 48202. E-mail: czxu@ece.eng.wayne.edu.*

Manuscript received 3 May 2008; revised 25 Jan. 2009; accepted 9 Mar. 2009; published online 17 Mar. 2009.
Recommended for acceptance by K. Hwang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-05-0166.
Digital Object Identifier no. 10.1109/TPDS.2009.51.

flexible, the load reassignment process incurs high overhead for changing IDs, especially in networks under churn.

Notice that the existing load balancing approaches assume that each node (or virtual node) has the same and constant DHT degree. That is, each node maintains the same number of neighboring relationships, irrespective of its capacity. The principle of power-law networks tells that the higher degree nodes tend to experience more query loads [2]. 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.

Recently, Castro et al. [7] exploited heterogeneity in congestion control by static mapping between node indegree and capacity and biasing high-capacity nodes for overlay neighbors. Static mapping cannot deal with nonuniform and time-varying file popularity and churn. Simple capacity bias may also make high-capacity nodes become bottlenecks. 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 the construction of capacity-aware DHTs. The indegrees are provably bounded.
- A policy for periodic indegree adaptation to deal with the nonuniform 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 ERT 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 4.1 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 [33], [12], [4]. "Virtual server" [33] 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 and Stoica [12] addressed the problem by arranging a real server for virtual ID space of consecutive virtual IDs. When a node selects its virtual servers, it first picks a random starting point and then selects one random ID within each of $\Theta(\log n)$ consecutive intervals of size $\Theta(1/n)$. In [4], 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. To achieve load balance, Byers et al. [6] suggested the direct application of the "power of two choices" paradigm, whereby an item is stored at the less loaded of two (or more) random alternatives.

Since a node with a longer interval has a higher probability of being contacted, the load balancing algorithms based on ID space interval assignment control traffic congestion due to the node heterogeneity in capacity. Initial key ID space partitioning is insufficient to guarantee load balance, especially in churn. It is often complemented by dynamic load reassignment. Godfrey and Rao et al. proposed schemes to rearrange load between heavy nodes and light ones according to their capacities so as to avoid bottleneck [11], [25]. They assumed that query load is uniformly distributed in the ID space.

Bharambe et al. [3] proposed a load balancing algorithm to deal with the congestion caused by biased lookups. They defined a node's load as the number of messages routed or matched per unit time. The algorithm proceeds in a way that heavily loaded nodes probe a number of sample nodes and requests lightly loaded nodes to leave from their current locations and rejoin at the location of the heavily loaded nodes. To get the load distribution information, each node periodically samples nodes within a certain distance and maintains approximate histograms. This requires much communication and maintenance cost, especially in churn. In addition, node ID changes due to the node leave/rejoin incurs high overhead.

The idea of using irregular routing tables with respect to the node capacity has been recently pursued by Hu et al. [14] and Li et al. [17]. Their foci were on a trade-off between maintenance overhead and lookup efficiency. Hu et al. proposed to deploy large routing tables in high-capacity nodes to exploit node heterogeneity and improve lookup efficiency. Li et al. designed an Accordian mechanism on Chord to vary the table size in different network scales and churn rates without compromising lookup efficiency. Chun et al. [9] evaluated the impact of neighbor selection on performance and resilience of structured P2Ps. The neighbor selection takes into consideration different types of heterogeneity, such as node capacity and network proximity. With node capacity consideration, a node chooses its neighbors which have the smallest processing delay. However, this method likely makes high-capacity neighbors become bottlenecks. In fact, always biasing the query routing toward high-capacity nodes exacerbate the problem if these nodes do not have enough capacity to handle a sudden query flow caused by churn. The query load should

be assigned to high-capacity nodes as well as low-capacity nodes proportional to their capacities.

Castro et al. [7] proposed a neighbor selection algorithm to construct routing tables based on the node's different 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. Moreover, ERT mechanism handles more than node heterogeneity. It handles biased lookups and churn by adjusting table indegree and out-degree 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 [24], [18], [2], [19], [8] for recent examples. Like congestion control in DHT networks, their solutions are based on the principle of power-law networks that high-degree nodes play an important role in communication.

Osokine [24] proposed a reactive flow control mechanism, in which receivers drop packets when becoming overloaded and senders infer the likelihood that a neighbor will drop packets based on the responses that they receive from the neighbor. This mechanism is acceptable when queries are flooded across the network, because even if a node drops a query, other copies of the query will propagate through the network. However, it is not suitable for random walks search [18]. Lv et al. [18] revealed that in power-law networks, such as Gnutella, the high-degree nodes often experience high query load. They suggested that P2P systems should adopt graph building algorithms that reduce the likelihood of very-high-degree nodes. On the contrary, Adamic et al. [2] took advantage of the feature of power-law networks that high-degree nodes play an important role in communication. They proposed a message-passing algorithm, which forwards queries to high-degree nodes so as to speed up the process of finding targeted files. It is noticed that high-degree nodes are not necessarily high-capacity nodes and that they are prone to bottleneck if they carry an extremely large share of query traffic. To overcome this disadvantage and to consider node heterogeneity, Lv et al. [19] proposed query flow control and topology adaptation algorithms to let higher capacity nodes have a higher degree, and forward queries to these nodes. The algorithms gradually change the overlay topology, based on the status of each neighbor link, so that queries flow toward the nodes that have sufficient capacity to handle them. However, the bias to high-capacity nodes may also make them to become hot spots. Chawathe et al. [8] proposed an active flow control scheme, which acknowledges the existence of heterogeneity and adapts to it by assigning flow-control tokens to nodes base on available capacity. In the scheme, a node distributes k flow-control tokens means that the node is willing to accept k queries, and a sender is allowed to direct a query to a neighbor only if it has a flow-control token of the neighbor.

Because of DHTs' strictly controlled topology and precisely defined routing algorithm, their routing table neighbors are restricted and query routing path is fixed. Since, the flow control algorithms 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.

TABLE 1
Notations

Notation	Description
n	the number of nodes
$\tilde{n}$	estimated n
c	node capacity
$\tilde{c}$	estimated c
l	node load
s	query distribution share
d	node indegree
α	indegree per unit capacity
d^{∞}	node maximum indegree
β/μ	a pre-defined percentage
$\gamma_c/\gamma_n/\gamma_l$	a factor
T	a unit time period
ν_{max}	the maximum incoming query rate
ν_{min}	the minimum incoming query rate
g	congestion rate

3 Elastic DHT

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

3.1 Definitions

We assume an DHT network with n physical nodes, labeled as an integer from 1 to n. Node $i, 1 \leq i \leq n$, has a capacity that the node 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. In order to facilitate the description, we define notions used in this paper in Table 1.

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, in fair load distribution, each node's load share is 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. In this case, a node's capacity is fully utilized and it is also not overloaded. 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 network. In [19], Lv et al. showed that a high-degree node in Gnutella network would most likely experience high query load. We apply the 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 an DHT network without churn, l_i is directly related to d_i. New policies will be proposed to deal with the nonuniformly distributed nodes, file queries, and churn in Sections 3.3 and 4.1.

It is known that the indegree of a node is determined by the number of outlinks of other nodes. In order to have a node's indegree to be proportional to its capacity, we reverse the relationship to determine node's outdegree by setting an appropriate value of indegree. To the end, we need to address two technical questions:

1. How to determine a node's indegree in order to make full use of its capacity and keep it light loaded at the same time?
2. How to construct ERTs with different indegrees of nodes, and meanwhile retaining the original DHT routing table function for lookup routing?

We normalize node capacity c so that the average of c is 1; that is, $\sum_i c_i = n$. Recall that in a uniform system without churn, l_i is related to d_i directly. It follows that

$$s_i \approx \frac{d_i / \sum_i d_i}{c_i / \sum_i c_i} \text{ and } d_i \approx c_i \frac{\sum_i d_i}{n}$$

when $s_i = 1$ ideally. Taking $\frac{\sum_i d_i}{n}$ 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 and query initiation rate in a nonuniform system with churn. High-load system should have small alpha while low-load system could have large alpha. Considering that the maximum number of queries a node can process at a time depends on its capacity, we define node i's maximum indegree d_i^∞ as $\lfloor 0.5 + \alpha c_i \rfloor$ since α is indegree per unit capacity. In order to keep the decimal fraction, 0.5 is used. The initial indegree of node i is βd_i^∞, where β is a predefined percentage for reservation purpose. It is determined according to system actual query load. If nodes are easily overloaded due to the query load, β should be a small value. Otherwise, it could be set to a large value. There is a trade-off in α determination: if α is too small, high-capacity nodes cannot be fully utilized because of low indegree, while a large α makes it very possible that low-capacity nodes, even high-capacity nodes, become heavy nodes. Moreover, large α leads to extra maintenance cost for overlay connections. Therefore, it is important to determine a suitable α. In the following, we present an initial indegree assignment algorithm for the construction of ERT.

3.2 Initial Indegree Assignment

Like bidirectional links in Gnutella, we assume that each DHT node i maintains a backward outlink (*backward finger*) for each of its inlink, in order to know the nodes which forward queries to it. Consequently, a double link is maintained for each routing table neighbor. Once node i joins the system, it needs to build its routing table based on DHT protocols. In order to control each 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. Each neighbor in node i's routing table creates a backward finger to node i. After building a basic routing table, node i then probes other nodes which 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.

Node (10100000)	
Routing table	
1	(1010000-1)
2	(101000-1-0)
3	(10100-1-00)
4	(1010-1-000)
5	(101-1-0000)
6	(11-0-00000)
7	(1-1-100000)

(a)

Node (10100000)	
Routing table	
1	(1010000-1)
2	([101000-1-0, 101000-1-1])
3	([10100-1-00, 10100-1-11])
4	([1010-1-000, 1010-1-111])
5	([101-1-0000, 101-1-1111])
6	([11-0-00000, 11-0-11111])
7	([1-1-100000, 1-1-1111111])

(b)

Fig. 1. Examples of Chord routing table without and with loose restriction.

To probe nodes for indegree expansion, the IDs of those nodes should be first decided. The ID set can be determined in the opposite way of the original DHT neighbor selection algorithm. We will explain it later in DHT examples. After the determination of the ID set, node i sends requests targeting to some of the nodes in the set that are not in the list of its backward fingers. On receiving such a request, the node which is responsible for the ID checks if it can take node i as its routing table neighbor. If can, it adds node 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. In the following, we take Chord, Cycloid, Pastry, and Tapestry as DHT examples to explain indegree expansion algorithm. The algorithm can be applied to other structured overlays that have little flexibility in the selection of neighbors by relaxing their routing table neighbor constraints.

In Chord, a node has $O(\log n)$ fingers in its routing table. We represent ID of node i as i. Then, the $(m+1)$th finger of node i is the successor of $(i + 2^m)(0 \leq m \leq d-1)$. Fig. 1a shows the routing table of the node with ID (10100000). Actually, Chord's routing table neighbor constraint can be loosed. That is, the $(m+1)$th finger is a set of successors succeeding the successor of $(i + 2^m)(0 \leq m \leq d-1)$. Fig. 1b illustrates the resultant routing table of node (10100000). Such that node i can ask nodes with a set of predecessor IDs of $(i - 2^m)(0 \leq m \leq d-1)$ to point to it. For example, assume node i's ID is (1010-1-011) and it can send requests targeting to ID $\in [1010-0-000, 1010-0-011]$ to take it as their 4th routing table finger. If the ID of node j which receives the request is $\in [1010-0-000, 1010-0-011]$, node j takes i as its 4th routing table finger. Otherwise, node j cannot take i as its 4th routing table finger.

Cycloid is a constant-degree DHT where each node has seven outdegree. We keep the Cycloid topology but remove the restriction of the constant degree. That is, a node's outdegree can be any value. 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 $(k-1, a_{d-1}a_{d-2}\ldots \overline{a}_k x x \ldots x)$ 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. Fig. 2 shows an example of routing table of Cycloid node with

Node (4,101-1-1010)
Routing table
Cubical neighbor: (3,101-0-xxxx)
Cyclic neighbor: (3,101-1-1100)
Cyclic neighbor: (3,101-1-0011)

Fig. 2. An example of Cycloid routing table.

ID (4,101-1-1010). 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 $(k+1, a_{d-1}a_{d-2}\ldots \overline{a}_k xx\ldots x)$ to ask nodes to take it as their cubical neighbors, and also it can send requests targeting $(k+1, a_{d-1}a_{d-2}\ldots a_k xx...x)$ to ask nodes to take it as their cyclic neighbors. For instance, node i (3,101-0-0000) can probe $(4, 101-1-xxxx)$ to increase its indegree. Let's say, node i first sends a request targeting (4,101-1-0000). Assuming node j receives the request, if $j \in (4, 101-1-xxxx)$, j adds i as its cubical neighbor and node i builds a backward finger to j. If node i needs to increase its indegree to 10, but it is only 6 after cubical backward finger probing, node i probes cyclic backward finger for the rest 4 indegree. Algorithm 1 shows the pseudocode of indegree expansion algorithm in Cycloid. In the pseudocode, we represent the cubical ID $a_{d-1}a_{d-2}\ldots a_k\ldots a_0$ in node ID $(k, a_{d-1}a_{d-2}\ldots a_k\ldots a_0)$ as a_{id}.

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 $ID = (k+1, a_{d-1}a_{d-2}\ldots\overline{a}_k xx\ldots x)$
3: $id = (k+1, a_{d-1}a_{d-2}\ldots\overline{a}_k 00\ldots 0)$
4: **while** not finish probing all IDs in ID $\wedge((d_i^\infty - d_i) \geq \beta d_i^\infty)$ **do**
5: **while** id is in backward fingers **do**
6: $id = (k+1, a_{id}++) \in ID$
7: **end while**
8: probe ID for cubical neighbor inlink
9: $id = (k+1, a_{id}++) \in ID$
10: **end while**
11: *//probe backward fingers of cyclic neighbor*
12: figure out ID of cyclic neighbor inlinks $ID = (k+1, a_{d-1}a_{d-2}\ldots a_k xx\ldots x)$
13: $id = (k+1, a_{d-1}a_{d-2}\ldots a_k 00\ldots 0)$
14: **while** not finish probing all IDs in ID $\wedge((d_i^\infty - d_i) \geq \beta d_i^\infty)$ **do**
15: **while** id is in backward fingers **do**
16: $id = (k+1, a_{id}++) \in ID$
17: **end while**
18: probe ID for cyclic neighbor inlink
19: $id = (k+1, a_{id}++) \in ID$
20: **end while**

Pastry's routing table is organized into $\lceil \log_{2^b} n \rceil$ (b is a configuration parameter) rows with $2^b - 1$ entries each. An entry at row m of node i's routing table refers to a node whose ID shares the node i's ID in the first m digits, but whose $(m+1)$th digit is not the $(m+1)$th digit in node i's ID. For example, node (10233102) can have nodes with ID $(10xxxxxx)$ at its row 2, as shown in Fig. 3. Tapestry's routing table neighbor selection algorithm is similar to Pastry's. Since each entry has multiple choices, node $i(a_{d-1}a_{d-2}\ldots a_{k-1}a_k\ldots a_0)$ can send request targeting to $(a_{d-1}a_{d-2}\ldots a_{k-1}\overline{a}_k x\ldots x)$ to ask nodes to take it as their kth row entry. Algorithm 2 shows the pseudocode of indegree assignment algorithm.

Node (10233102)			
Routing table			
(-0-2212102)	1	(-2-2301203)	(-3-1203203)
0	(1-1-301233)	(1-2-230203)	(1-3-021022)
(10-0-31203)	(10-1-32102)	2	(10-3-23302)
(102-0-0230)	(102-1-1302)	(102-2-2302)	3
(1023-0-322)	(1023-1-000)	(1023-2-121)	3
(10233-0-01)	1	(10233-2-32)	
0		(102331-2-0)	
		2	

Fig. 3. An example of Pastry routing table.

Algorithm 2. Pseudo-code for indegree assignment algorithm of node i join

1: **while** (node i's routing table has not created) **do**
2: probe another node for routing table neighbors
3: **if** (receive j's posiRpy) **then**
4: set j as routing table neighbor, i.e., create a outlink to j
5: **end if**
6: **end while**
7: **while** $(d_i^\infty - d_i \geq \beta d_i^\infty)$ **do**
8: execute indegree expansion algorithm
9: **if** receive k's posiRpy **then**
10: creates backward finger to k
11: d_i++
12: **end if**
13: **end while**
{**node j execution:**}
14: **if** (receive node i's neighbor probing) **then**
15: **if** $(d_j^\infty - d_j \geq 1)$ **then**
16: send posiRpy to i
17: build a backward finger to i
18: d_j++
19: **else**
20: send negRpy to i
21: **end if**
22: **end if**
{**node k execution:**}
23: **if** (receives node i's backward finger probing) **then**
24: **if** (i can be k's routing table neighbor) **then**
25: create outlink to i
26: send posiRpy to i
27: **end if**
28: **end if**

The initial indegree assignment algorithm proceeds repeatedly. We prove that the ERT indegree resulted from the algorithm is bounded. We assume an DHT that manages

a unit-size ID space, i.e., $[0,1) \subseteq \mathbb{R}$ employing arithmetic modulo 1, and the DHT uses consisting hash [15] 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 [20], [23] for details of such an estimation process. We denote $\tilde{n}$ as estimated n and $\tilde{c}$ as estimated c.

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.*

Proof. With γ_c and γ_n as the maximum error factor of a node's estimated capacity and $n, \tilde{c}_i$ is within the factor γ_c of c_i and $\tilde{n}$ is within γ_n of n w.h.p. Thus, the indegree first assigned to node i is at most $\lfloor 0.5 + \tilde{c}_i\alpha(\tilde{n})\rfloor \le \tilde{c}_i\alpha(\tilde{n}) + O(1) \le \gamma_c c_i \alpha(\gamma_n n) + O(1) \le \gamma_c c_i \alpha(n) + O(1)$. The indegree first assigned to node i is at least $\lfloor \tilde{c}_i\gamma(\tilde{n}) - 0.5\rfloor \ge \tilde{c}_i\alpha(\tilde{n}) - O(1) \ge c_i/\gamma_c\alpha(n/\gamma_n) - O(1) \ge c_i/\gamma_c\alpha(n) - O(1)$. □

Algorithm 3. Pseudo-code for periodic indegree adaptation algorithm of node i

```
1: if (g_i > γ_l) //node i overloaded? then
2:     ask ½(l_i − c_i)α nodes pointed by backward fingers to
       delete i form their routing tables
3:     if (receive k's posiRpy) then
4:         delete backward finger to k
5:         d_i − −
6:         d_i^∞ + +
7:     end if
8: else
9:     if (g_i < 1/γ_l) then
10:        probe ½(c_i − l_i)α nodes for backward finger
11:        if (receive k's posiRpy) then
12:            create backward finger to k
13:            d_i + +
14:            d_i^∞ − −
15:        end if
16:    end if
17: end if
```

3.3 Periodic Indegree Adaptation

In practice, nodes join and leave DHT overlays continuously and the files in the system may have nonuniform and time-varying popularity. 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. To ensure that queries flow toward nodes with sufficient capacity, the congestion control protocol should adapt to the change of query rate and lookup skewness caused by nonuniform and time-varying file popularity, as well as network churn.

We design a 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 tables, then deletes corresponding backward fingers, and decreases its maximum indegree d_i^∞ correspondingly. μ is a predefined percentage. To choose a backward finger to remove, it chooses the one with the longest logical distance. In the case with the same logical distances, it chooses the one with the longest physical 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. Algorithm 3 shows the pseudocode of periodic indegree adaptation algorithm.

The following theorem shows that the ERT indegree in the process of adaptation remains 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.*

Proof. Node i does not need to update its indegree when $\tilde{c}_i\gamma_l \ge l_i \ge \tilde{c}_i/\gamma_l$, where $\frac{c_i}{\gamma_c} \le \tilde{c}_i \le \gamma_c c_i$. Assume that during a certain time period T, the average incoming query rate per inlink of node i is ν_i; that is, on average, there are ν_i queries coming from each inlink during time T. Assume that node i has degree d_i at a certain time point during T such that $l_i = \nu_i d_i$.

When $l_i > \tilde{c}_i\gamma_l$, node i updates its indegree to $d_i - \mu(l_i - \tilde{c}_i)$. Since $l_i > \tilde{c}_i\gamma_l, \nu_i d_i > \tilde{c}_i\gamma_l$, the indegree is at most $d_i - \mu\tilde{c}_i(\gamma_l - 1), d_i - \mu\frac{c_i}{\gamma_c}(\gamma_l - 1)$. It is $d_i - \mu(\nu_i d_i - \tilde{c}_i) \ge (1-\mu)\nu_i d_i + \frac{\mu c_i}{\gamma_c}$. Consequently, the indegree is decreased to between $(1-\mu)\nu_i d_i + \frac{\mu c_i}{\gamma_c}$ and $d_i - \mu\frac{c_i}{\gamma_c}(\gamma_l - 1)$. On the other hand, when $l_i < \frac{c_i}{\gamma_l}$, node i updates its indegree to $d_i + \mu(\tilde{c}_i - l_i)$. Since $l_i < \frac{\tilde{c}_i}{\gamma_l}, \nu_i d_i < \frac{\tilde{c}_i}{\gamma_l}$, the indegree is at least $d_i + \mu\tilde{c}_i(1 - \frac{1}{\gamma_l}), d_i + \mu\frac{c_i}{\gamma_c}(1 - \frac{1}{\gamma_l})$. It is $d_i + \mu(\tilde{c}_i - \nu_i d_i) \le (1 - \mu\nu_i)d_i + \mu c_i\gamma_c$. Therefore, the indegree is increased to between $d_i + \mu\frac{c_i}{\gamma_c}(1 - \frac{1}{\gamma_l})$ and $(1 - \mu\nu_i)d_i + \nu c_i\gamma_c$. The indegree is changed until it reaches a status that $\tilde{c}_i\gamma_l \ge \nu_i d_i \ge \tilde{c}_i/\gamma_l, \frac{c_i\gamma_c\gamma_l}{\nu_{min}} \ge d_i \ge \frac{c_i}{\gamma_c\gamma_l\nu_{max}}$. □

For example, in a network of size 2,048, if a node's capacity is 50 and its average incoming query rate is 0.5, its indegree is bounded by 100 in the case $\gamma_l = 1$. The following theorem shows that the outdegree of an ERT is bounded as well. We leave its proof to the Appendix.

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

4 Dynamic Randomized Query Forwarding

Periodic indegree adaptation may not be sufficient to deal with query load imbalance. In this section, we present a complementary randomized query forwarding algorithms to help forward queries toward light nodes so as to further reduce lookup latency.

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

4.1 Query Forwarding Policies

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 $i = (4,101\text{-}1\text{-}1010)$ has cubical outlinks pointing to nodes (3,1010-0000), (3,1010-0001), and (3,1010-0010). For an incoming query destined for its cubical neighbors based on the 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 a 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. For example, node i receives a query with key (2,1010-0011) and the query should be forwarded to a cubical neighbor according to Cycloid routing algorithm. It first randomly chooses two options (3,1010-0010) and (3,1010-0001) among the three cubical neighbors if $b = 2$. If most of a node's neighbors are heavily loaded, the forwarding policy cannot improve the performance too much. Although such situation may not occur frequently, in this case, the node will generate more neighbors to handle this problem based on the periodic indegree adaption algorithm.

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 [31], [30] 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 (see Section 4.2 for justification). 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 [21] proved that granting a task with two server choices and dispatching it to one of the servers with less workload leads 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. [22] improved the performance of two-choice method 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 adapt 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 nodes i and j. It then selects the better one, say node i, and remember the least loaded node between i and j after node i increases by one load unit. We assume that the node is still i, which is used for the next query 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 node i, it repeats the process again.

To further reduce the heavy nodes in query routings, a query flows by the use of the information of overloaded nodes encountered before, to avoid overloaded node in the succeeding routings. For example, a lookup node has three options for forwarding a query: i_1, i_2, and i_3. Using the above method, it forwards the query to i_2 with information of overloaded node i_1. In the second step, node i_2 has three options: i_1, j_1, and j_2. Because it knows that i_1 is overloaded, it will not select i_1 as a option for query forwarding. Algorithm 4 shows the pseudocode of the topology-aware randomized query forwarding algorithm.

Algorithm 4. Pseudo-code for topology-aware randomized query forwarding algorithm executed by node i

```
1: receive query Q with overloaded node information A
2: determine the set of outlinks for the query forwarding
   based on DHT routing algorithm.
3: choose options J = {j_1, j_2 ...} from the outlink set
   excluding overloaded node in A
4: if memory has a node j_a then
5:    randomly choose a node j_b from J
6: else
7:    randomly choose two nodes j_a and j_b from J
8: end if
9: //choose the better node from j_a and j_b
10: probe node j_a and j_b for load status
11: if j_a and j_b are heavy then
12:  add j_a and j_b to A
13:  forward Q and A to the least heavily loaded node
14: else
15:  if one node is light and one node is heavy in j_a and j_b
     then
16:     add the heavy node to A
17:     forward Q and A to the light node
18:  end if
19: else
20:     choose nodes J_log logically nearest to target ID from
        j_a and j_b
21:     choose nodes J_phy physically nearest to node i from
        J_log
22:     forward Q and A to a node in J_phy
23: end if
```

4.2 Analysis of Query Forwarding Policies

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 neighbors. 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 Poisson distributed [33].

The forwarding model can be regarded as a variation of *strong threshold supermarket model* (STSM) proposed in [21] if we take γ_l as the threshold in the latter. In the STSM, customers arrive at a Poisson 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 predetermined threshold. If both choices are over the threshold, the customer 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 STSM but heterogeneous in the QFM.

Along the line of analytical approach in [21], we analyze the performance of the randomized query forwarding algorithms. The following theorem shows that the two-way randomized query forwarding policies improve lookup efficiency exponentially over random walking. Readers are referred to the Appendix for the proof.

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 properties of the ERT-based congestion control protocol through simulation built on an $O(1)$-degree Cycloid network. ERT can also be applied to other DHT networks. Simulations on other $O(\log n)$-degree networks are expected to produce better results. We assumed a bounded Pareto distribution for the capacity of nodes [11]. This distribution reflects real-world situations where machines' capacities vary by different orders of magnitude. Recall that the maximum indegree of a node, say node i, is defined in Section 3.2 as $\lfloor 0.5 + \alpha c_i \rfloor$, where $c_i = nc_i / \sum_i c_i$ is the normalized capacity of the node. That is, the node can handle this amount of queries at one time. We define node i's load as the number of queries in its query processing queue. If node i has more than $\lfloor 0.5 + \alpha c_i \rfloor$ queries in its queue, it is overloaded. We further assumed that the queries be generated according to a Poisson process at a rate of one per second [33], with a random source node and a random target key, unless otherwise noted. Table 2 lists the parameters of the simulation and their default values.

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 rate should be kept around 1, implying the node is neither overloaded nor under utilized, and its capacity is fully 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. Recall that share

$$s_i = \frac{l_i / \sum l_i}{c_i / \sum c_i}.$$

  It represents the performance of fair load distribution, i.e., the total system load is distributed among nodes based on their capacity. The objective of fair sharing is hard to achieve in DHT networks because of a number of reasons. First, it is hard to collect the load and capacity of other nodes. Second, DHT is a dynamic system with continuous node joins and departures, as well as continuous query initialization. It is hard to control instant share of each node in such a dynamic situation. Third, since query load is not uniformly distributed among the nodes, it changes with file popularity and churn. Although fair sharing is not the objective of congestion control, 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 in order to reduce lookup latency.

TABLE 2
Simulated Environment and Algorithm 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 adaptation constant μ	$1/2$
Indegree adaptation 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 also include the results due to a "virtual server" load balancing method [12] (VS) and a neighbor selection algorithm for indegree control (NS) [7]. The NS algorithm bears resemblance to the ERT initial indegree assignment as to select neighbor based on node indegree bound. However, NS always selects high-capacity nodes as neighbors. It may overcompromise the needs of low-capacity nodes. ERT makes full use of node capacity by letting nodes reach their indegree bounds. Moreover, ERT allows dynamic indegree adaptation (A) and facilitates query forwarding (F) to deal with network

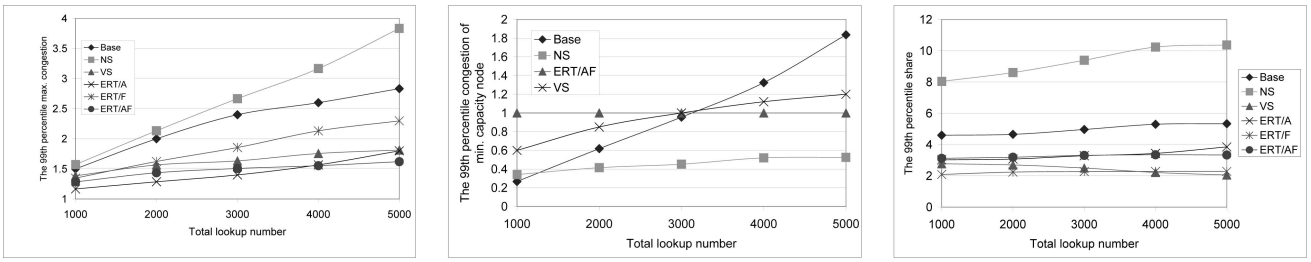


Fig. 4. Effectiveness of congestion controls. (a) Maximum congestion, (b) congestion of minimum capacity node, and (c) share.

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 1,000 to 5,000, with 1,000 increase in each step. We also varied the processing time of a query in a light node from 0.1 to 2.1 second and five 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. Fig. 4a shows the congestion rate due to each method increases as more lookup queries arrive. The NS protocol produces a higher 99th percentile maximum congestion 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 neighbors. The high-capacity nodes may turn out to be overloaded.

In contrast, VS and ERT/AF lead to much lower congestion rates. That is, they are more effective in controlling the load of each node based on its capacity. Also, ERT/AF outperforms VS, especially when the system is highly loaded. The relative performance between NS, VS, and ERT/AF, as shown in Fig. 4a, can be verified by the 99th percentile congestion rate of minimum capacity node in Fig. 4b. It is expected to see that the low-capacity node becomes congested as the query load increases. Without congestion control (Base), the congestion rate increases sharply. The congestion control protocols delay the occurrence of congestion. In particular, the NS protocol overprotected 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.

To further evaluate the impact of individual factors of adaptation and forwarding, we include the results due to ERT/A and ERT/F in Fig. 4a. The figure shows that indegree adaptation (ERT/A) reduces the congestion rate of Base significantly in various load conditions and performs consistently better than VS. Forwarding (ERT/F) alone may not work as well as VS in reducing the congestion rate. In ERT/F, the congestion rate would grow rapidly as the lookup number increases. It implies that forwarding is effective in controlling node congestion when query load is light, but becomes less effective as the system load increases.

Fig. 4c 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 Figs. 4a and 4b. That is, NS heavily relies on high-capacity nodes for query routing. Excluding low-capacity nodes in neighbor selection may lead to a waste of system resources because their capacities can be used to ease the burden of high-capacity nodes in certain situations. In contrast, 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. An ideal share in DHT is difficult to achieve because of the DHT strict controlled topology, routing algorithm, nonuniform and variable file popularity, and churn. VS approximates fair sharing by static ID space assignment. However, it is at the cost of more maintenance overhead and lookup cost. It cannot handle skewed lookups either.

From Fig. 4c, we can also see that forwarding (ERT/F) alone leads to a lower share rate than ERT/AF. It is expected because query forwarding that forwards queries only to nodes with sufficient capacity could reach a balanced load distribution in concept. In the next section, we will see it is at the cost of lookup efficiency.

5.2 Lookup Efficiency

Lookup latency is determined by two factors: lookup path length and query processing time in each node along the path. Fig. 5a shows the total number of overloaded nodes encountered in query routings grows with the query load. It also shows that ERT/AF leads to much high lookup efficiency in comparison with the others. Although 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. 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. The fixed outdegree of nodes in NS and VS prevents each node from adapting traffic load on nodes elastically. In contrast, ERT/AF enables each node to match its indegree to its capacity and adapt its indegree in response to the change of its experienced query load. Furthermore, the query forwarding operation helps avoid overloaded nodes during

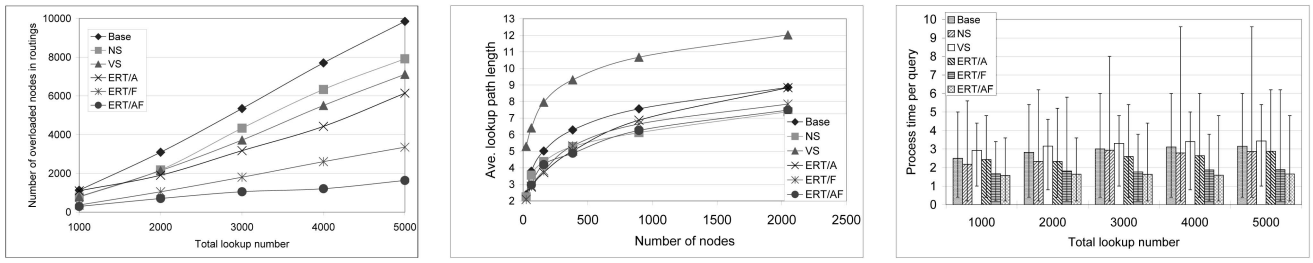


Fig. 5. Effectiveness of congestion control protocols on lookup efficiency. (a) Heavy nodes in routings, (b) lookup path length, and (c) lookup time.

query routing, leading to higher lookup efficiency. From the figure, we can also observe that both adaptation (ERT/A) and forwarding (ERT/F) lead to a significant reduction of overloaded nodes in comparison with Base, NS, and VS. In another word, the effectiveness of ERT/AF in avoiding overloaded nodes in routings is attributed to combined effects of adaptation and forwarding algorithms.

Fig. 5b 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 [12] that VS achieves the objective of load balancing at the cost of lookup efficiency, and the path length increases by at most an additive constant. In contrast, ERT/A and ERT/F reduce the path lengths of Base. It implies that both indegree adaptation and forwarding contribute to the effectiveness of ERT/AF in lookup path length reduction. Adaptation offers each node more neighbor candidates to forward a query. The locality-aware forwarding algorithm takes both logical and physical distance into account in decision making. It always chooses the neighbors that are logically closest to the destination and then physically closest to the destination. Thus, the forwarding algorithm greatly reduces the lookup path length. Both indegree adaptation and forwarding algorithms reduce the lookup path length, leading to much shorter lookup path length of ERT/AF. Likewise, NS considers node distance in neighbor selection and reduces the lookup path length of Base significantly.

Fig. 5c shows the average, 1st, and 99th percentiles of processing time per query as combined effects 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 dramatically reduces the processing time per query of VS and Base. In ERT/AF, periodic indegree adaptation tunes each node degree to its load adaptively, and the forwarding operation tends to direct queries to light nodes that have sufficient capacity to handle them promptly. Both of the adaptation (ERT/A) and forwarding (ERT/F) factors help improve query processing efficiency in ERT/AF by reducing congested nodes and lookup path length. The efficiency seems attributed more to the forwarding factor.

5.3 Indegree of Nodes and Maintenance Cost

A node with a higher indegree would most likely experience higher query load and vice versa. In order to illustrate the query load imbalance due to indegree variance, we measured the number of nodes with different indegrees in Cycloid DHT with different dimensions. In a Cycloid DHT, the nodes can be divided to two groups: low-indegree nodes and high-indegree nodes. The low-indegree nodes have indegree equals to 5, and the high-indegree nodes have indegree equals to 14, 16, 18, 20, and 22 when the dimension equals to 6, 7, 8, 9, and 10, respectively. Fig. 6 illustrates the results. We can observe that there always have a number of high-indegree nodes. These nodes constitute 10-15 percent of the total nodes. This means that these nodes will receive more queries and, hence, experience higher query load than low-indegree nodes. The experiment results confirm the existence of query load imbalance.

Recall that ERT-based congestion control protocol achieves its goal using elastic routing tables to adapt each node indegree to its load. In addition to maintain the tables, each node needs to maintain a list of backward fingers (the same number of its indegree). We measured the maximum indegree and outdegree of each node and calculated the average, 1st, and 99th percentiles of these values in each method. As we mentioned that the node degree is fixed in NS and VS, but is variable in ERT. We use maximum indegree and outdegree instead of average for the evaluation of the management overhead of ERT in the worse case. Figs. 7a and 7b plot the results. Although inlinks in Base and VS don't need to be maintained, we include their degrees for comparison. As expected, the figures show that the degree 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 indegrees to adapt to different query load accordingly, some lightly loaded node indegrees reach a high value to request more load. Indegree change leads to

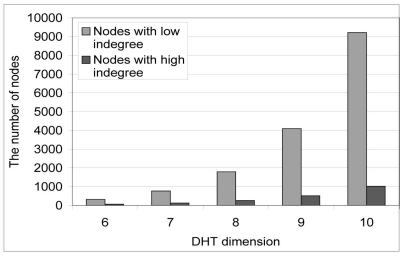


Fig. 6. Indegrees of nodes.

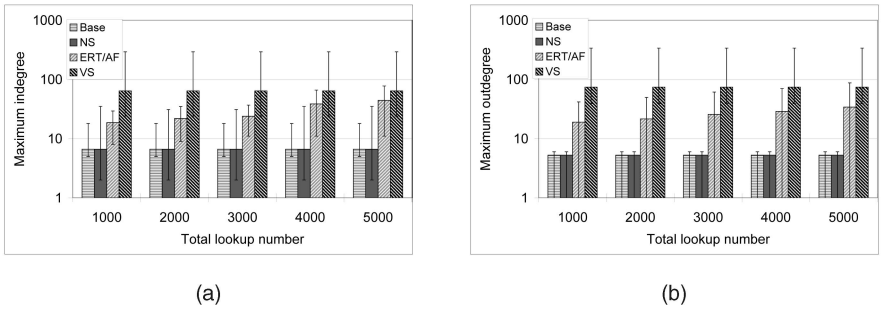


Fig. 7. Degrees of routing tables in different congestion control protocols. (a) Indegree and (b) outdegree.

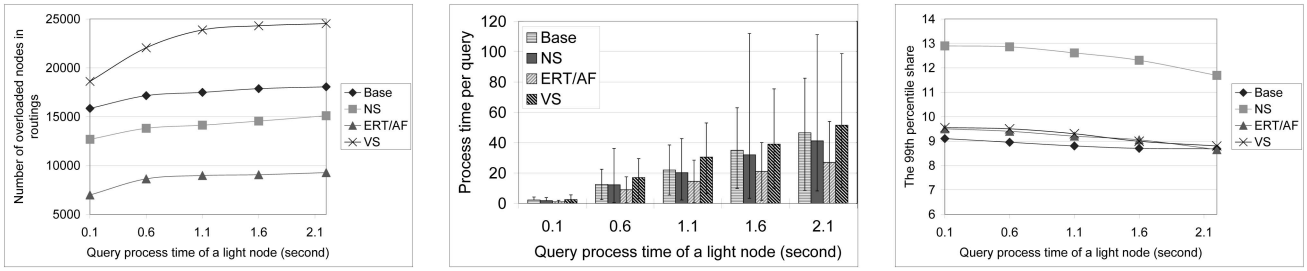


Fig. 8. Effectiveness of congestion control protocols in skewed lookups. (a) Heavy nodes in routings, (b) lookup time, and (c) share.

outdegree change. The indegree and outdegree of VS are much higher than others because that virtual node usage leads to larger overlay size. Our results turn out that the combination of the average, 1st, and 99th percentiles of 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 ERT only needs a little extra maintenance cost.

5.4 Effect of Skewed Lookup

Besides node heterogeneity in capacity, query load imbalance occurs with nonuniform and time-varying file popularity and peer interest variation. In this section, we consider the effect of skewed lookups.

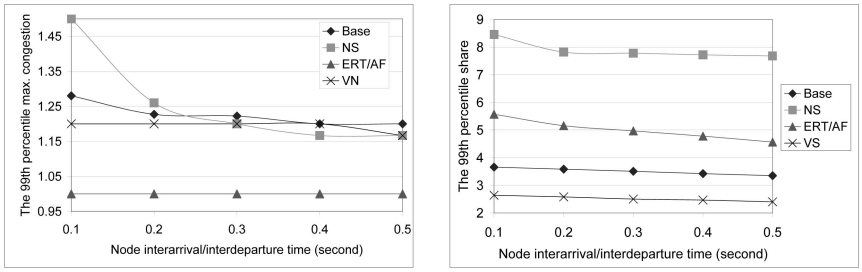
We consider an "impulse" of 100 nodes whose IDs are distributed over a contiguous interval of the ID space, and whose interests are in the same 50 keys randomly chosen from the ID space. We varied the query process rate from 0.1 to 2.2 second per query on a light node, with 0.5 second increase in each step. Figs. 8a and 8b plot the number of overloaded nodes in routings and the query processing time of each method, respectively. It is surprising to see that these result values of VS are much more than Base. As claimed by the authors in [12] 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, a real node selects IDs of its virtual nodes randomly within consecutive intervals. When query load concentrates on a certain ID space interval, the load is allocated to consecutive virtual servers. Since most of the virtual servers 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 topology-aware randomized forwarding algorithm, ERT/AF 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.

Fig. 8c plots the 99th share of each method. By comparing it with Fig. 4c, 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 nodes in neighbor selection. It is a resource waste to let low-capacity nodes idle.

5.5 Effect of Churn

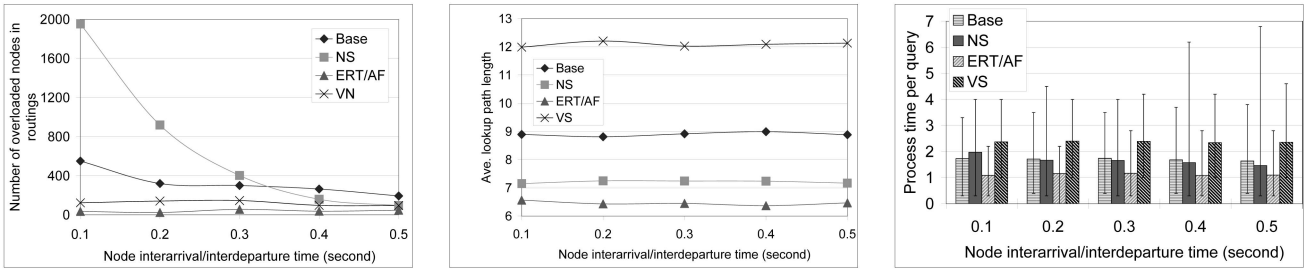
In DHT networks with churn, a great number of nodes join, leave, and fail continually and rapidly, leading to continuous change of overlay topology. This gives another challenge to congestion control. This section evaluates ERT/AF's adaptability to different levels of churn. In this experiment, the lookup rate was modeled by a Poisson process with a rate of 1; that is, there was a lookup every 1 second. The node join/departure rate was also modeled 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. Our results are collected from all node including the current nodes in the system when all lookups complete and the nodes departed.

Fig. 9a shows the 99th percentile maximum congestion of each method. Comparing it with Fig. 4a, we find that the rate of each method in churn is lower than without churn at the point of 3,000 lookups. It is because with continuous nodes join, the same query load is distributed among more nodes than in static DHT. The rates of NS and Base grow inversely proportional to node interarrival time, and the rates of VS and ERT/AF maintain constant. When node interarrival/interdeparture time is 0.1 second, the rate of



(a) (b)

Fig. 9. Effectiveness of congestion control protocols in networks with churn. (a) Maximum congestion and (b) share.



(a) (b) (c)

Fig. 10. Effectiveness of congestion control protocols on lookup efficiency in churn. (a) Heavy nodes in routings, (b) lookup path length, and (c) lookup time.

NS is higher than Base's, and it decreases slightly below the Base's when node interarrival time is 0.3-0.5 seconds. This implies that NS has difficulty to cope with high churn. Recall that in NS, high-capacity nodes have denser inlinks. In high churn, some high-capacity nodes may don't have enough capacity for a sudden query flow, which originally should be responsible by nodes departed. In a modest churn, the flow is not so intense for nodes to handle. 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/AF keeps the rate close to 1 in different levels of churn, in controlling node congestion in churn.

Fig. 9b 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 99th percentile share of ERT/AF 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 99th percentile share is higher than Base, and VS has more balanced query load distribution than others.

Fig. 10a shows the number of heavy nodes 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 Fig. 9a. It confirms that ERT/AF performs the best in reducing heavy nodes processing query. Fig. 10b shows the lookup path length of each method. Comparing it with Fig. 5b, we can detect that there's no big difference, except that ERT/AF has less path length in churn. We also recorded average timeout for each method. A time-out occurs when a node tries to contact a departed node. The average time-out of ERT/AF is 0 and is less than 0.06 in other approaches. The reason for shorter path lengths and less time-outs of ERT/AF is that its ERT avoids time-outs by letting each node have multiple neighbors in each table entry. Consequently, when a entry neighbor left, others can be used as a substitute instead of making a detour routing. Fig. 10c shows the average, 1st, and 99th percentiles of query processing time per node of each method. They are consistent to those without churn in Fig. 5c, except that NS yields higher latency than Base in high churn. It validates the conclusion that NS is not efficient in coping with churn.

5.6 Effect of Adaptation and Query Forwarding

To evaluate the quality of the indegree adaptation and topology-aware randomized query forwarding, we compare ERT/AF with algorithms without indegree adaptation (ERT/F) or without query forwarding algorithm (ERT/A). Fig. 4a plots the 99th percentile maximum congestion rates of different versions. From the figure, we can observe that the topology-aware two-way randomized forwarding algorithm is effective in reducing the congestion rate of Base when query load is not high, but becomes not so effective when the system is highly loaded. For this reason, it is imperative to have a complementary method to guarantee low node congestion. The figure shows that the indegree adaptation algorithm reduces the congestion rate significantly in various load conditions, which implies the dramatic contribution of ERT/A in controlling node congestion. The 99th percentile shares in Fig. 4c confirm the superior performance of forwarding. The query forwarding algorithm controls query flow to light nodes, ensuring that queries are forwarded only to nodes with sufficient capacity to handle them. By adjusting node indegree adaptively, indegree adaptation algorithm

also helps for fair load balancing, though the improvement is not so much as forwarding.

Fig. 5a shows the heavy node number encountered in each lookup path. From the figure, we can observe that both forwarding and indegree adaptation greatly help eliminate heavy nodes. Their combination demonstrates an accumulated effect. Fig. 5b plots the path length of each version. Recall that topology-aware randomized query forwarding algorithm takes node logical distance and physical distance into account in routing. It is expected that forwarding leads to a short lookup path. By providing multiple neighbor candidates in each step for query routing, indegree adaptation is also effective in reducing path length in most cases. Overall, the combined effect on overloaded nodes reduction and lookup path length shortening results in a great saving of lookup latency, as shown in Fig. 5c. The figure shows that both of ERT/A and ERT/F reduce processing time per query of Base, and their effectiveness are aggregated together, leading to much higher query processing efficiency of ERT/AF. From the observations, we can conclude that both of ERT/A and ERT/F play an important role in ERT/AF in congestion control and lookup latency reduction. Lack of either one will degrade the final effectiveness of ERT/AF.

6 Conclusions

DHT networks have an inherent congestion problem caused by query load due to the nature of heterogeneity and dynamism of nodes. Nonuniform and time-varying file popularity makes the problem more severe. 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 while control each node's load below its capacity. It improves the lookup efficiency in DHT network by reducing lookup latency.

Appendix

Proof of Theorem 3.3. A node i with cyclic index k_i can have at most 2^{k_i} IDs for routing table entry selection. We use $\mathcal{O}_i$ to represent a set of these IDs, and it can have at most 2^{k_i} IDs for routing table entry backward finger selection. We use $\mathcal{I}_i$ to represent a set of these IDs. Because there are totally $d' \cdot 2^{d'}$ IDs and n nodes in a Cycloid system with dimension d', the probability of node number for an ID is $\frac{n}{d'2^{d'}}$, denoted by η. Consequently, the number of nodes in 2^{k_i} IDs is proximately $\eta 2^{k_i}$, denoted by J_i. We define average ID space responsible by a node as I, and use I_i to represent the I of node i.

Let X_i be the indicator variable for the event that node i probes node j for indegree expansion. If j is chosen by i, then i has a backward finger to node j, which increments j's outdegree with a outlink to i. Our purpose is to find out the upper bound of $\sum_i X$ represented by X. We assume that Cycloid nodes also probe their successors and predecessors for indegree expansion. The probability that node j is chosen to have an outlink pointing to i is $\frac{1}{J_i}$. The probability that node j is within $a/2$ number of i's successors or predecessors is $aI + 2I_j$. Let's assume that node i probes m_i IDs. Then, the probability is expressed as $P(|ID_j - ID_i| \leq \max\{0, m_i - J_i\}I) = \max\{0, m_i - J_i\}I + 2I_j \cdot P(m_i > J_i)$.

In the case that $i \in \mathcal{O}_j$, when $J_i \geq m_i, E[X_i] = m_i \frac{1}{J_i}$; when $J_i < m_i, E[X_i] = J_i \frac{1}{J_i} + (m_i - J_i)I + 2I_j$. In the case that $i \notin \mathcal{O}_j$, when $J_i \geq m_i, E[X_i] = 0$ and when $J_i < m_i, E[X_i] = (m_i - J_i)I + 2I_j$. Therefore,

$$E[X_i] = \begin{cases} min\{J_i, m_i\}\dfrac{1}{J_i} + \max\{0, m_i - J_i\}I \\ \quad + 2I_j P(m_i > J_i), & i \in \mathcal{O}_j; \\ \max\{0, m_i - J_i\}I + 2I_j P(m_i > J_i), & i \notin \mathcal{O}_j \wedge i \neq j. \end{cases}$$

$$\begin{aligned} E[X] &= \sum_{i \in R} E[X_i] \\ &= \sum_{i \in \mathcal{O}_j} \Big(min\{J_i, m_i\}\frac{1}{J_i} + \max\{0, m_i - J_i\}I \\ &\quad + 2I_j P(m_i > J_i) \Big) + \sum_{i \notin \mathcal{O}_j \wedge i \neq j} (\max\{0, m_i - J_i\}I \\ &\quad + 2I_j P(m_i > J_i)) = \sum_{i \in \mathcal{O}_j} \Big(min\{J_i, m_i\}\frac{1}{J_i} \Big) \\ &\quad + \sum_{i \in R \wedge i \neq j} (\max\{0, m_i - J_i\}I + 2I_j \cdot P(m_i > J_i)) \\ &= \frac{1}{\eta 2^{k_{j-1}}} \sum_{i \in \mathcal{O}_j} min\{J_i, m_i\} + \sum_{i \in R \wedge i \neq j} (max\{0, m_i - \eta 2^{k_i}\}I \\ &\quad + 2I_j \cdot P(m_i > J_i)) \\ &\leq \frac{1}{\eta 2^{k_{j-1}}} \eta 2^{k_j} \max_{i \in \mathcal{O}_j} \mathcal{I}_i + I_{max} \sum_{i \in R \wedge i \neq j} \big(m_i - \eta 2^{K_i} + 2\big) \\ &\leq \frac{2\gamma_c \gamma_l c_{max}}{\nu_{min}} + \gamma_n \Big[O(1) - O\Big(\frac{2^{d'}}{d'} \Big) \Big] \\ &\leq \frac{2\gamma_c \gamma_l c_{max}}{\nu_{min}} - O\Big(\frac{2^{d'}}{d'} \Big) + O(1). \end{aligned}$$

□

We define $b_i(t)$ as the number of servers with i spare capacities at time t; $m_i(t)$ as the number of servers with at most i spare capacities at time t; $p_i(t) = d_i(t)/d$ as the fraction of servers of i spare capacity; and $s_i(t) = m_i(t)/d$ as the fraction of servers with at most i spare capacities. Such that $p_i = s_i - s_{i-1}$. In an *empty system*, which corresponds to one with no customers, $s_c = 1$, and $s_i = 0$ for $i < c$. A *fixed point* π is a point p in which $\frac{ds_i}{dt} = 0$.

The rate of spare capacity changing in a node depends on whether it has more or fewer than threshold, T, spare

capacities. In the following, we calculate $\frac{ds_i}{dt}$ in the case $i \geq T-1$ and $i < T-1$, respectively. An arriving query occupies the ith capacity of a server if one of b events happen: first, its first choice has $i+1$ spare capacities; second, its first choice has $\leq T-1$ spare capacities and its second choice has $i+1$ spare capacities; ..., its first b-1 choices have $\leq T-1$ spare capacities and its bth choice has $i+1$ spare capacities. So that, there are $\lambda n(p_{i+1} + s_{T-1}p_{i+1} + s_{T-1}^2 p_{i+1} + \cdots + s_{T-1}^{b-1} p_{i+1})$ servers whose spare capacities change from $i+1$ to i during dt. Meanwhile, dp_i servers change their spare capacities from i to $i+1$. As a result, we get

$$\frac{ds_i}{dt} = \lambda\left(p_{i+1} + s_{T-1}p_{i+1} + s_{T-1}^2 p_{i+1} + \cdots + s_{T-1}^{b-1} p_{i+1}\right) - p_i,$$
$$i \geq T-1, \quad \frac{ds_i}{dt} = \lambda(s_{i+1} - s_i)\frac{s_{T-1}^b - 1}{s_{T-1} - 1} - (s_i - s_{i-1}),$$
$$i \geq T-1$$

When $i < T-1$, the number of queries arriving over dt is $\lambda b dt$, and that for an query being forwarded to a server with $i+1$ spare capacity is $b_i dt = d(s_i - s_{i-1})dt$. Consequently, $\frac{ds_i}{dt} = \frac{1}{b} \cdots \frac{dm_i}{dt} = \lambda(s_{i+1}^b - s_i^b) - (s_i - s_{i-1})$. The differential equations for the query forwarding model (QFM) when $i < T = 1$ considering a node with b specific neighbors is $\frac{ds_i}{dt} = \lambda(s_{i+1}^b - s_i^b) - (s_i - s_{i-1}), i < T-1$.

We get the differential equations for QFM:

$$\frac{ds_i}{dt} = \begin{cases} \lambda(s_{i+1} - s_i)\dfrac{s_{T-1}^b - 1}{s_{T-1} - 1} - (s_i - s_{i-1}), & c \geq i \geq T-1; \quad (3) \\ \lambda\left(s_{i+1}^b - s_i^b\right) - (s_i - s_{i-1}), & i < T-1. \quad (4) \end{cases}$$

Lemma A.1. *The QFM with $d \geq 2$ has a unique fixed point with $\sum_{i=c-1}^{-\infty} s_i < \infty$ given by*

$$\begin{cases} s_i = (\lambda - A)\dfrac{A^{c-i} - 1}{A - 1} + A^{c-i}, \\ \quad A = \lambda\dfrac{s_{T-1}^b - 1}{s_{T-1} - 1}, & T-1 \leq i \leq c; \\ s_i = \lambda\dfrac{b^{T-i-1} - 1}{b - 1} \cdot s_{T-1}^{b^{T-i-1}-1}, & i < T-1. \end{cases}$$

Proof. With the condition $\frac{ds_i}{dt} = 0$ for all i, we derive the value of s_i including s_{T-1} when $c \geq i \geq T-1$ with $s_c = 1$. We summer the (3) over all if $c \geq i \geq T-1$, and get $s_{c-1} = \lambda - A + As_c$, assuming $A = \lambda\frac{s_{T-1}^b - 1}{s_{T-1} - 1}$. By induction: $s_{c-2} = (\lambda - A)(1 + A) + A^2 \cdots$, we get

$$s_i = (\lambda - A)\frac{A^{c-i} - 1}{A - 1} + A^{c-i}, \quad A = \lambda\frac{s_{T-1}^b - 1}{s_{T-1} - 1},$$
$$c \geq i \geq T-1.$$

By summing the (4) over all $i \leq T-1$ with $s_{-\infty} = 0$, we derive that $s_{T-2} = \lambda s_{T-1}^b$. By induction:

$$s_{T-3} = \lambda s_{T-2}^b = \lambda\left(\lambda s_{T-1}^b\right)^b = \lambda^{\frac{b^2-1}{b-1}} s_{T-1}^{b^2} \ldots,$$

we get $s_i = \lambda^{\frac{b^{T-i-1}-1}{b-1}} \cdot s_{T-1}^{b^{T-i-1}-1} (i < T-1)$. We use $\sum_{i=c-1}^{-\infty} s_i < \infty$ to ensure that the sum converges absolutely. □

Proof of Theorem 4.1. By (4), we can get that in the case when $i < T-1$, an incoming query arriving on a node at time t lets the node has i spare capacity with probability $s_{i+1}(t)^b - s_i(t)^b$, and this query becomes the $(c-i)$th query in the process waiting queue of the server. Therefore, the expected waiting time of the query is:

$$\sum_{i=T-2}^{-\infty} (c-i)\left(s_{i+1}(t)^b - s_i(t)^b\right) = (c - T + 2)s_{T-1}^b + \sum_{i=T-2}^{-\infty} \left(s_i^b(t)\right)$$

By (3), we can get that in the case when $i \geq T-1$, the expected waiting time of a query is

$$\sum_{c-1}^{T-1} (c-i)(A/\lambda)(s_{i+1}(t) - s_i(t)) = (A/\lambda)\left(\sum_{i=c}^{T} s_i - (c - T + 1)s_{T-1}\right), \quad A = \lambda\frac{s_{T-1}^b - 1}{s_{T-1} - 1}.$$

By Lemma A.1, at $t \to \infty$, the QFM converges to the fixed point. So that the expected waiting time for a query in a server can be made:

$$(A/\lambda)\left(\sum_{i=c}^{T}\left((\lambda - A)\frac{A^{c-i} - 1}{A - 1} + A^{c-i}\right) - (c - T + 1)s_{T-1}\right) + (c - T + 2)s_{T-1}^b + \sum_{i=T-2}^{-\infty}\left(\lambda^{\frac{b^{T-i}-b}{b-1}} s_{T-1}^{b^{T-i}-b}\right) + o(1).$$

In QFM, the time a query waits on a node when $i \geq T-1$ is less than the time when $i < T-1$ because in the former case the query is processed by a light node. The above bound can be enlarged to $\sum_{i=c}^{-\infty} s_i^b = \sum_{i=c-1}^{-\infty} \lambda^{\frac{b^{c-i}-b}{b-1}}$. Then we can apply the proved result of exponential time improvement in [21] to QFM. □

ACKNOWLEDGMENTS

This research was supported in part by US National Science Foundation grants CNS-0834592, CNS-0832109, CCF-0611750, MCS-0624849, CNS-0702488, CNS-0834592, and CNS-0832109. An early version of this work [29] was presented in the Proceedings of ICDCS '06.

REFERENCES

[1] Mute, http://mute-net.sourceforge.net/, 2009.
[2] L.A. Adamic, B.A. Huberman, R.M. Lukose, and A.R. Puniyani, "Search in Power Law Networks," *Physical Rev. E*, vol. 64, pp. 46135-46143, 2001.
[3] A.R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *Proc. ACM SIGCOMM*, 2004.
[4] M. Bienkowski, M. Korzeniowski, and F.M. auf der Heide, "Dynamic Load Balancing in Distributed Hash Tables," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS)*, 2005.
[5] S. Bono et al., "Mantis: A Lightweight, Server-Anonymity Preserving, Searchable P2P Network," technical report, Johns Hopkins Univ., 2004.
[6] J. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," *Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS)*, 2003.

[7] M. Castro, M. Costa, and A. Rowstron, "Debunking Some Myths About Structured and Unstructured Overlays," *Proc. Second Conf. Symp. Networked Systems Design & Implementation (NSDI),* 2005.
[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella Like P2P Systems Scalable," *Proc. ACM SIGCOMM,* 2003.
[9] B.G. Chun, B.Y. Zhao, and J.D. Kubiatowicz, "Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks," *Proc. Fourth Int'l Workshop Peer-To-Peer Systems (IPTPS),* 2005.
[10] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science,* vol. 2009, pp. 46-66, Springer, 2001.
[11] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," *Performance Evaluation,* vol. 63, no. 3, pp. 217-240, 2006.
[12] B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," *Proc. IEEE INFOCOM,* 2005.
[13] P. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP),* 2003.
[14] J. Hu, M. Li, W. Zheng, D. Wang, N. Ning, and H. Dong, "SmartBoa: Constructing P2P Overlay Network in the Heterogeneous Internet Using Irregular Routing Tables," *Proc. Third Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2004.
[15] D. Karger et al., "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," *Proc. 29th Ann. ACM Symp. Theory of Computing (STOC),* 1997.
[16] B. Levine and C. Shields, "Hordes: A Multicast-Based Protocol for Anonymity," *J. Computer Security,* vol. 10, no. 3, pp. 213-240, 2002.
[17] J. Li et al., "Bandwidth Efficient Management of DHT Routing Tables," *Proc. Second Symp. Networked System Design and Implementation (NSDI '05),* 2005.
[18] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. Ann. ACM Int'l Conf. Supercomputing (ICS),* 2001.
[19] Q. Lv et al., "Can Heterogeneity Make Gnutella Scalable?" *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2002.
[20] G. Manku, "Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables," *Proc. 23rd Ann. ACM Symp. Principles of Distributed Computing (PODC),* 2004.
[21] M. Mitzenmacher, "On the Analysis of Randomized Load Balancing Schemes," *Proc. Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA),* 1997.
[22] M. Mitzenmacher et al., "Load Balancing with Memory," *Proc. 43rd IEEE Symp. Foundations of Computer Science (FOCS),* 2002.
[23] S. Nath, P.B. Gibbons, S. Seshan, and Z.R. Anderson, "Synopsis Diffusion for Robust Aggregation in Sensor Networks," *Proc. Second ACM Conf. Embedded Networked Sensor Systems (SenSys),* 2004.
[24] S. Osokine, "The Flow Control Algorithm for the Distributed 'Broadcast-Route' Networks with Reliable Transport Links," technical report, 2001.
[25] A. Rao et al., "Load Balancing in Structured P2P Systems," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2003.
[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM,* pp. 329-350, 2001.
[27] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proc. Middleware Conf.,* 2001.
[28] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking (MMCN),* 2002.
[29] H. Shen and C. Xu, "Elastic Routing Table with Provable Performance for Congestion Control in DHT Networks," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS),* 2006.
[30] H. Shen and C. Xu, "Hash-Based Proximity Clustering for Load Balancing in Heterogeneous DHT Networks," *Proc. 20th Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* 2006.
[31] H. Shen and C. Xu, "Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 6, pp. 849-862, June 2007.
[32] H. Shen, C. Xu, and G. Chen, "Cycloid: A Scalable Constant-Degree P2P Overlay Network," *Performance Evaluation,* vol. 63, no. 3, pp. 195-216, 2006.
[33] I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. on Networking,* vol. 11, no. 1, pp. 17-32, Feb. 2003.
[34] B.Y. Zhao et al., "Tapestry: An Infrastructure for Fault-Tolerant Wide Area Location and Routing," Technical Report No. UCB/CSD-01-1141, 2001.

Haiying Shen received the BS degree in computer science and engineering from Tongji University, China, in 2000, and the MS and PhD degrees in computer engineering from Wayne State University in 2004 and 2006, respectively. She is currently an assistant professor in the Department of Electrical and Computer Engineering, and the director of the Pervasive Communications Laboratory of Clemson University. Her research interests include distributed and parallel computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, wireless networks, resource management in cluster and grid computing, and data mining. Her research work has been published in top journals and conferences in these areas. She was the program cochair for a number of international conferences and member of the Program Committees of many leading conferences. She is a member of the IEEE, the IEEE Computer Society, and the ACM.

Cheng-Zhong Xu received the BS and MS degrees from Nanjing University in 1986 and 1989, respectively, and the PhD degree from the University of Hong Kong in 1993. He is a professor in the Department of Electrical and Computer Engineering of Wayne State University (WSU) and the director of the Center for Networked Computing Systems. His research interest includes networked computing systems and applications, in particular scalable and secure Internet services and architecture, scheduling and resource management in distributed, parallel, and embedded systems, and autonomic systems management for highly reliable computing. He has published more than 140 peer-reviewed scientific papers in archival journals and conferences in these areas. He is the author of "*Scalable and Secure Internet Services and Architecture*" (Chapman & Hall/CRC Press, 2005) and the leading coauthor of "*Load Balancing in Parallel Computers: Theory and Practice*" (Kluwer Academic, 1996). He serves on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *Journal of Parallel and Distributed Computing*, *Journal of Parallel, Emergent, and Distributed Systems*, *Journal of Computers and Applications*, and *Journal of High Performance Computing and Networking*. He has also guest edited special issues for several other journals on network services and security in distributed systems. He has served a number of international conferences and workshops in various capacities as program chair, general chair, and plenary speaker. He was a recipient of "President's Award for Excellence in Teaching" of WSU in 2002 and "Career Development Chair Award" in 2003. He is a senior member of the IEEE and the IEEE Computer Society, and a member of the ACM.