Low Traffic Overlay Networks with Large Routing Tables

Chunqiang Tang†, Melissa J. Buco†, Rong N. Chang†, Sandhya Dwarkadas‡
Laura Z. Luan†, Edward So†, and Christopher Ward†

ABSTRACT

The routing tables of Distributed Hash Tables (DHTs) can vary from size $O(1)$ to $O(n)$. Currently, what is lacking is an analytic framework to suggest the optimal routing table size for a given workload. This paper (1) compares DHTs with $O(1)$ to $O(n)$ routing tables and identifies some good design points; and (2) proposes protocols to realize the potential of those good design points.

We use total traffic as the uniform metric to compare heterogeneous DHTs and emphasize the balance between maintenance cost and lookup cost. Assuming a node on average processes 1,000 or more lookups during its entire lifetime, our analysis shows that large routing tables actually lead to both low traffic and low lookup hops. These good design points translate into one-hop routing for systems of medium size and two-hop routing for large systems.

Existing one-hop or two-hop protocols are based on a hierarchy. We instead demonstrate that it is possible to achieve completely decentralized one-hop or two-hop routing, i.e., without giving up being peer-to-peer. We propose 1h-Calot for one-hop routing and 2h-Calot for two-hop routing. Assuming a moderate lookup rate, compared with DHTs that use $O(\log n)$ routing tables, 1h-Calot and 2h-Calot save traffic by up to 70% while resolving lookups in one or two hops as opposed to $O(\log n)$ hops.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Design, Management, Performance

Keywords

Peer-to-Peer System, Overlay Network, Distributed Hash Table

1. INTRODUCTION

In recent years, Distributed Hash Tables (DHTs) have been proposed as the infrastructure for building a wide range of distributed applications such as storage [2], content distribution [4], and search engines [21]. A DHT organizes nodes into a structured overlay network and can efficiently map a key to the node that is responsible for the key through distributed routing. The designs of DHTs vary dramatically. Early designs [20] use small $O(\log n)$ routing tables, due to the concern that big routing tables are hard to maintain and cannot scale to large systems. Later designs use $O(\sqrt{n})$ [8] or even $O(n)$ [7] routing tables and argue that it is feasible to do so.

This paper provides an analytic framework to suggest the optimal routing table size for a given workload. A workload is parameterized by a tuple $<n, l, f>$, where n is the number of nodes in the system, l is the average node lifetime, and f is the average number of lookups that a node processes per second (i.e., the node is the destination of the lookups). We use *traffic* as the uniform metric to compare heterogeneous DHTs with $O(1)$ to $O(n)$ routing tables. Our analysis shows that the most traffic-efficient routing size is proportional to $O(fl\ln(n))$.

Our analysis does have practical use. It helps us to identify pitfalls in existing DHT designs that are mainly driven by the desire to improve lookup latency, e.g., the argument [7] that it is favorable to maintain $O(n)$ routing tables for systems with millions of nodes. Our analysis shows that it is not cost-effective to do so for systems larger than a few thousand nodes. Otherwise, it could introduce 1,000 times more traffic than traditional DHTs [20].

Most existing DHTs are intended for environments similar to those for peer-to-peer file sharing systems such as Gnutella and KaZaA, and hence are designed to handle a high churn rate, assuming node lifetimes as short as several minutes [17]. Consequently, they argue for small $O(\log(n))$ routing tables, which seems reasonable as both node lifetime l and lookup rate f are low. However, DHTs are inherently unsuitable for environments with a high churn rate because DHTs mandate data placement on nodes; by contrast, a node in Gnutella stores its own data locally. In DHTs, when a node joins, some data must be copied to that node; when the node leaves, data stored on that node must be copied to another node. Even if the routing tables can be maintained correctly under a high churn rate [17], the high traffic due to data movement would render the system unusable [1].

Not surprisingly, most deployed DHT applications [4, 22] run on relatively stable but unreliable nodes. Open DHT [22] is one prominent example. After several years of extensive research on DHTs, Open DHT is perhaps the only deployed DHT running at a large scale. It runs on PlanetLab and offers services to nodes outside the DHT, including mobile nodes. We believe that this model is the future of DHT. It is unnecessary and inefficient to include every node that uses the DHT services as part of the DHT. If a node lives for only several minutes, the overhead caused by the join and leave of the node and related data movement is likely to dwarf the services, if any, provided by the node during its short lifetime. Selecting only good quality nodes to provide services can result in a DHT that is smaller, faster, and more efficient. Even KaZaA [10] uses just a subset of super nodes to provide lookup services.

†IBM T. J. Watson Research Center, Hawthorne, NY 10532. {ctang, mbuco, rong, luan, edwardso, cw1}@us.ibm.com.
‡Computer Science Department, University of Rochester, Rochester, NY, 14627-0226. sandhya@cs.rochester.edu.

SIGMETRICS'05, June 6–10, 2005, Banff, Alberta, Canada.

When a DHT is provided as a service to nodes outside the DHT, we assume that each DHT node on average processes 1,000 or more lookups during its entire lifetime, i.e., $fl \geq 1000$. For instance, assuming a 2.9 hour node lifetime (the average node lifetime in Gnutella [19]), each DHT node needs to process one lookup every 10 seconds; assuming a one week node lifetime, each DHT node needs to process one lookup every 600 seconds. We believe this assumption $fl \geq 1000$ is reasonable. If the lookup rate is extremely low, then the DHT is underutilized. The architect should downsize the DHT to reduce unnecessary overheads and resource wastes, resulting in increased lookups submitted to each DHT node.

Under the assumption $fl \geq 1000$, our analysis shows that large routing tables with several hundred to one thousand entries actually lead to both low traffic and low lookup hops. This design point translates into one-hop routing (with $O(n)$ routing tables) for systems with up to a few thousands nodes; or two-hop routing (with $O(\sqrt{n})$ routing tables) for systems with up to a few million nodes.

One-hop and two-hop routings are efficient in both traffic and lookup hops, but their large routing tables are hard to maintain. Existing proposals for one-hop or two-hop routing are either hierarchical [5, 7, 8, 15, 18]—in which nodes have different roles and the load is unevenly distributed—or assume a particular query distribution that limits its generality [16]. We will demonstrate that it is possible to achieve one-hop or two-hop routing without giving up being peer-to-peer. A peer-to-peer architecture has many good properties such as resilience and load balance, which are the reasons that originally motivated DHTs [20].

We propose what we believe are the first practical non-hierarchical protocols for one-hop routing (*1h-Calot*) and two-hop routing (*2h-Calot*). Compared with traditional DHTs that use $O(\log n)$ routing tables, 1h-Calot and 2h-Calot save total traffic by up to 70% while resolving lookups in one or two hops as opposed to $O(\log n)$ hops. Their fast lookups are particularly attractive for interactive applications such as search engines [21] and name resolution.

To maintain the large routing tables in a scalable fashion, 1h-Calot and 2h-Calot multicast node arrivals and departures through $O(n)$ different trees embedded in the overlay. The "trees" in 1h-Calot and 2h-Calot are conceptual and require no explicit maintenance. 2h-Calot's randomized algorithm further exploits virtual nodes running on the same computer to route among remote nodes in a purely peer-to-peer fashion. Both 1h-Calot and 2h-Calot are extremely simple: multicast maintains the routing tables; information in the routing tables is then used to guide multicast and routing.

The remainder of the paper is organized as follows. Section 2 compares heterogeneous DHTs in order to identify the good design points. Sections 3 and 4 present the design and analysis of our one-hop and two-hop protocols, respectively. Section 5 evaluates our protocols through extensive simulation. Related work is discussed in Section 6. Section 7 concludes the paper.

2. OPTIMAL ROUTING TABLE SIZE

Previous works [6, 12, 13, 23] mainly used resilience and lookup latency as the metrics to compare DHTs with $O(\log(n))$ routing tables. Instead, we use total traffic (both maintenance and lookup) as the metric to compare DHTs with $O(1)$ to $O(n)$ routing tables under a strawman model. Our goal is to reveal the fundamental impact of routing table size on the traffic of DHTs. Traffic is relevant because a low-traffic DHT allows the architect to use a smaller and faster DHT to handle a given load. In general, DHTs with larger routing tables introduce higher maintenance traffic but have fewer routing hops and hence lower lookup traffic. A good design should strike a balance between them to minimize the total traffic.

We assume that node lifetime follows an exponential distribution

$$p_l(t) = \lambda_l e^{-\lambda_l t}, \tag{1}$$

where $\lambda_l = \frac{1}{l}$ and l is the average node lifetime. We assume that node arrival is a Poisson process with rate λ_e. The probability that k nodes join during a time period t is

$$P(X = k) = \frac{(\lambda_e t)^k}{k!} e^{-\lambda_e t}. \tag{2}$$

To maintain a stable population of n nodes with an average lifetime l, the node arrival rate $\lambda_e = n\lambda_l = \frac{n}{l}$. We assume the lookups that a node processes follow a Poisson process with rate f.

Both lookup messages and messages for routing table maintenance are small. The payload typically includes a DHT key and the IP address of a node. Unless otherwise noted, we assume communications use UDP/IP; lookup and maintenance messages have unit size s (including both packet header and payload); the messages are explicitly acknowledged and the acknowledgments have size $0.5s$. Our analysis ignores packet loss and retransmissions at the network layer. We assume that the targets of lookups distribute uniformly across all nodes. We assume an "ideal" representative for each category of DHTs in order to shed light on the fundamentals. When it comes to a specific DHT design, we also consider other factors such as resilience and lookup hops. Sections 3 and 4 will address more realistic implementation issues. Below, we use the total traffic metric to compare DHTs with $O(1)$ to $O(n)$ routing tables.

Degree-Diameter Optimal DHTs

For a network with n nodes in which each node has d neighbors (i.e., the node degree is d) , the network's diameter D (maximum hops of the shortest paths between any two nodes) is bounded [13] by: $D \geq \lceil \log_d(n(d-1)+1) \rceil - 1$. We refer to DHTs that approach this lower bound as degree-diameter optimal DHTs [9, 11, 13, 14]. At the abstract level, DHTs with the same node degree introduce similar maintenance traffic but those with optimal diameters introduce lower lookup traffic. Our comparison therefore focuses on degree-diameter optimal DHTs with routing tables of different sizes. We use de Bruijn graphs [13] as the representative, in which a lookup on average takes $r \approx \log_d n$ hops.

We calculate the *minimal traffic*[1] needed to update the routing tables of a de Bruijn graph in the face of node arrivals and departures. In an n-node system with a node lifetime l, on average $\frac{n}{l}$ nodes join and $\frac{n}{l}$ nodes leave each second. When a node joins or leaves, at least one message is sent to notify each of its d routing neighbors, resulting in $\frac{n}{l}d$ messages for node arrivals and $\frac{n}{l}d$ messages for node departures. Furthermore, at least one message is needed to inform a new node of each of its d neighbors[2], resulting in $\frac{n}{l}d$ messages to set up the routing tables for new nodes.

Assuming all maintenance messages have unit size s and each is acknowledged by a packet of size $0.5s$, the maintenance traffic is

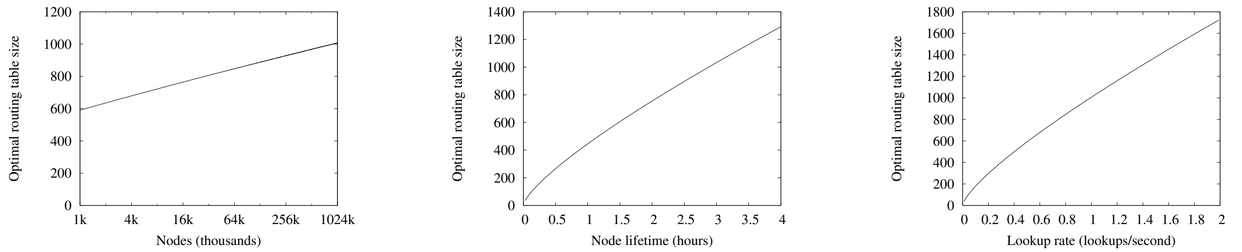
$$B_1 = (1 + 0.5)s \cdot (\frac{n}{l}d + \frac{n}{l}d + \frac{n}{l}d). \tag{3}$$

Each node processes f lookups per second, resulting in nf lookups in total. Each lookup takes $\log_d n$ hops in a de Bruijn graph. The traffic for lookups is therefore

$$B_2 = (1 + 0.5)s \cdot nf \log_d n. \tag{4}$$

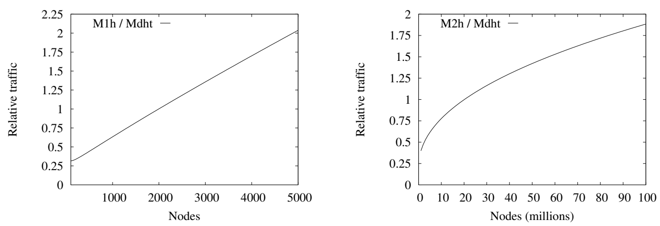
[1]In most existing DHTs, nodes probe their routing neighbors periodically. An "ideal" design can avoid this traffic. For instance, Calot uses overlay multicast to maintain routing tables.

[2]The traffic would be lower if the new node copies a complete routing table from an existing node in a single packet. This only affects our results by a very small constant factor. We choose not to consider this optimization here because it is adopted in few DHTs.



(a) f=1 lookup/second and l=2.9 hours. (b) n=1 million nodes and f=1 lookup/second. (c) n=1 million nodes and l=2.9 hours.

Figure 1: The optimal routing table size d that minimizes the total traffic (from Equation 6).



(a) One-hop schemes. (b) Two-hop schemes.

Figure 2: Traffic relative to traditional DHTs with $O(\log n)$ routing tables when l=2.9 hours and f=0.1 lookups/second. M_{1h}, M_{2h}, and M_{dht} are from Equations 7-9.

The total traffic (maintenance plus lookup) in a de Bruijn graph is

$$M = B_1 + B_2 = 1.5s \cdot (3\frac{n}{l}d + nf \log_d n). \tag{5}$$

We derive the routing table size d that minimizes the total traffic by setting the derivative of M with respect to d to 0.

$$\frac{\partial M}{\partial d} = 0 \implies d \ln^2 d = \frac{fl \ln n}{3} \tag{6}$$

The fl component in Equation 6 indicates that the optimal routing table size d is proportional to the number of lookups that a node processes during its entire lifetime. Previous comparisons mainly focused on the impact of node lifetime on system resilience, and ignored lookup rate. Our analysis instead shows that lookup rate is a critical parameter when designing DHTs for low traffic.

Equation 6 has no closed form solution. We solve it using Newton's method for a given workload $<n, l, f>$ and plot the results for some typical workloads in Figure 1. This figure shows that using large routing tables with several hundred to one thousand entries is actually efficient in traffic. This translates into one-hop routing (with $O(n)$ routing tables) for systems with up to a few thousand nodes, or two-hop routing (with $O(\sqrt{n})$ routing tables) for systems with up to a few million nodes. These are the good design points we focus on in Sections 3 and 4.

The above analysis makes some "ideal" assumptions: (1) when a node joins or leaves, this membership change can be efficiently disseminated to about 1,000 nodes; and (2) a node need not probe its 1,000 or so routing neighbors to maintain the accuracy of its routing table. These assumptions are obviously not met by existing solutions [13] based on a de Bruijn graph. Other systems that do use large routing tables are based on a hierarchy [5, 7, 8, 15, 18]. In Sections 3 and 4, we will present our peer-to-peer solutions.

One-hop Schemes

In one-hop schemes, nodes know each other: $d = n - 1 \approx n$. Substituting this into Equation 5, we obtain the total traffic

$$M_{1h} \approx 1.5s \cdot (3\frac{n^2}{l} + nf). \tag{7}$$

Two-hop Schemes

In ideal two-hop schemes, each node has $d = \sqrt{n}$ routing neighbors. Substituting this into Equation 5, we obtain the total traffic

$$M_{2h} = s(4.5\frac{n^{1.5}}{l} + 3nf). \tag{8}$$

Traditional DHTs

In traditional DHTs, each node has $O(\log n)$ routing neighbors and lookups are resolved in $O(\log n)$ hops. We consider an abstract version of the Chord protocol [20], in which each node has $d = \log_2 n$ neighbors and lookups on average take $\frac{\log_2 n}{2}$ hops.

Following the analysis process in Section 2, we know that the abstract Chord introduces traffic $M_{c1} = 4.5\frac{n}{l}s\log_2 n$ to update routing tables in the face of node arrivals and departures (see Equation 3 and note $d = \log_2 n$). In addition, each node sends a heartbeat message to each of its $\log_2 n$ neighbors every T=30 seconds. We assume the heartbeat messages have size $0.5s$. The traffic for heartbeats is $M_{c2} = \frac{0.5sn\log_2 n}{T}$. There are nf lookups in total. Lookups on average take $\frac{\log_2 n}{2}$ hops. The traffic for lookups is $M_{c3} = (1+0.5)s \cdot nf\frac{\log_2 n}{2}$. The coefficient 0.5 is because lookup messages are acknowledged. The total traffic therefore is

$$M_{dht} = M_{c1} + M_{c2} + M_{c3} = s \cdot n \log_2 n(\frac{4.5}{l} + 0.75f + \frac{0.5}{T}). \tag{9}$$

Comparing Traditional DHTs with Others

In Figure 2, we compare the traffic of traditional DHTs with that of one-hop schemes and two-hop schemes. Overall, the figure shows that one-hop and two-hop schemes can have low traffic and fast routing at the same time when fl is sufficiently high, for instance, in realistic DHTs like Open DHT [22]. In contrast to the argument [7] that it is favorable to maintain complete $O(n)$ routing tables for systems with up to a few million nodes, Figure 2(a) shows that one-hop schemes are only efficient for systems with up to several thousand nodes. With a few million nodes, a one-hop scheme could introduce 1,000 times more traffic than traditional DHTs. Figure 2(b) shows that an "ideal" two-hop scheme can be efficient for systems with up to millions of nodes. When the system has more than 20 million nodes, however, two-hop schemes introduce more traffic than traditional DHTs. Based on these observations, we propose our one-hop protocol (1h-Calot) for systems of medium size and two-hop protocol (2h-Calot) for large systems.

Route Caching and Reactive Maintenance

All the DHTs described above proactively maintain the accuracy of the routing tables. Another way to keep large routing tables is reactive maintenance, in which nodes cache other nodes they discovered in past lookups and reuse them in future lookups. There is no explicit maintenance operation. The drawback is that nodes may encounter frequent failures during lookups. Next, we calculate the probability of correct cache hit when nodes use their routing tables.

Suppose node N puts node S into its routing table when N discovers S through a lookup. The lookups that N issues follow a Poisson process with rate f. Assuming lookups are uniformly distributed, the lookups that N issues to target S is a Poisson process with rate $\lambda_v = f/n$ since there are n nodes. The interval between lookups from N to S follows an exponential distribution $p_v(t) = \lambda_v e^{-\lambda_v t}$. Node lifetime follows an exponential distribution $p_l(t) = \lambda_l e^{-\lambda_l t}$. When node N contacts node S at time x since the last lookup, the probability that S is still alive is $(1 - \int_{y=0}^{y=x} p_l(y)\ dy)$. The probability that node N finds node S alive when N issues a new lookup to S is therefore

$$P_{cache\_hit} = \int_{x=0}^{x=\infty} [\ p_v(x)\ (1 - \int_{y=0}^{y=x} p_l(y)\ dy)\ ]\ dx = \frac{\lambda_v}{\lambda_v + \lambda_l} = \frac{1}{1 + \frac{n}{lf}}. \quad (10)$$

Table 1 shows the cache hit rate P_{cache_hit} under typical workloads. When lookup rate f=0.1, about 49% of lookups fail on their first hop. A failed hop incurs a high latency as the query initiator has to wait for a long, conservative period before it timeouts. Since the cache hit rate is not sufficiently high, we consider reactive maintenance not suitable for interactive applications.

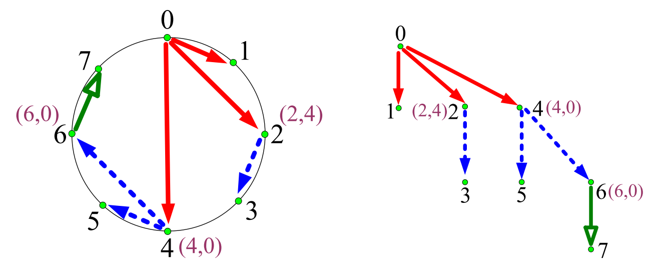
f	0.1	0.3	0.5	0.7	0.9
P_{cache_hit}	0.51	0.76	0.84	0.88	0.90

Table 1: Cache hit rate in Eq 10 (n=1,000 and l=2.9 hours).

3. 1H-CALOT FOR ONE-HOP ROUTING

The analysis in Section 2 shows that it is beneficial to use large routing tables. When implemented properly, they lead to both low traffic and low lookup hops. The challenge, however, is to efficiently maintain the large routing tables in the face of frequent node arrivals and departures. To this end, we propose *1h-Calot*. It uses overlay multicast to efficiently disseminate notifications for node arrivals and departures to all nodes. For systems with up to a few thousand nodes, 1h-Calot resolves lookups in one hop with a high probability while introducing traffic lower than traditional DHTs [20]. For larger systems, we will introduce in Section 4 our *2h-Calot* protocol that uses $O(\sqrt{n})$ routing tables. Unlike hierarchical one-hop or two-hop schemes [5, 7, 8, 15, 18], 1h-Calot and 2h-Calot are purely peer-to-peer.

Like Chord [20], 1h-Calot organizes nodes into a circular ring that corresponds to an identifier space [0, 2^{160}-1]. Each node is assigned an identifier by applying SHA-1 hashing to its IP address. We refer to a node's clockwise neighboring node along the ring as its *successor* and the counter-clockwise neighboring node as its *predecessor*. The predecessor node and the successor node of a *key* are defined similarly. Each object is associated with a key drawn from the identifier space, for instance, by applying SHA-1 hashing to the object's content. An object is stored on the node whose identifier is the closest to the object's key in absolute distance, regardless of the direction (clockwise or counter-clockwise).



(a) Multicast process in the overlay. (b) Multicast process as a tree.

Figure 3: Multicast tree for disseminating membership changes. This example uses a 3-bit identifier space. There are 8 nodes with identifiers 0-7. Node 0 just joined and acts as the root of the tree for announcing its arrival. Node 0 selects its finger nodes at exponentially increasing distance from itself as its children in the tree. Each child of node 0 is responsible for covering a range of the identifier space, for instance, the range $(2, 4)$ for node 2. The children of the root further select their finger nodes as their children to expand the tree, and so forth.

1h-Calot maintains a complete $O(n)$ routing table on every node. Ideally, nodes know each other and messages are delivered directly between the source and the destination. In the case that the routing tables are inaccurate (e.g., missing live nodes or listed dead nodes), routing may take longer. A node N always greedily forwards a lookup to the node P that is, to N's knowledge, the closest in absolute distance to the lookup key. If P is the right destination, the lookup is done. Otherwise, P further forwards the lookup to the node, to P's knowledge, closest to the destination, and so forth. If P is not responsive when N tries to forward P a lookup, N will timeout and try the second closest node. All communications in 1h-Calot use UDP and messages are explicitly acknowledged.

As in Chord, correct routing is guaranteed so long as each node correctly maintains its predecessor and successor (therefore a lookup always moves closer to its destination after each step). A node maintains its predecessor and successor through periodic heartbeat messages, but does not periodically probe any other node in its routing table. This is crucial to keep maintenance traffic low. Routing table maintenance is described in the next section.

3.1 Handling Node Joins and Leaves

We assume that a new node N knows through some out-of-band method about at least one node P already in the system. Node N copies a complete routing table from node P in order to have a global view of the system. Node N generates its identifier k by applying SHA-1 hashing to its IP address, and takes over objects that are closer to N from its predecessor and successor. Node N informs other nodes of its arrival by multicasting a notification through a tree rooted at N. The tree is implicitly embedded in the overlay (see Figure 3).

We first provide some definitions before describing the process for constructing the multicast tree. For a node V with identifier k, the *finger nodes* of node V are defined as the successor nodes of keys $r_i = k + 2^i$ $(i = 0, ..., 159)$. The finger nodes of node V distribute at exponentially increasing clockwise distance from V. As noted in Chord [20], with a high probability, each node has $O(\log_2 n)$ distinct finger nodes (note that, for example, the successors of keys r_0 and r_1 may be the same since r_0 and r_1 are close).

The new node N sits at the root of the multicast tree to announce its arrival. Among nodes in its routing table, node N selects its finger nodes as its children. Let S_i and s_i $(i = 1, \cdots, j)$ denote the j

finger nodes and their identifiers, respectively. Nodes S_i are ranked in increasing clockwise distance from N. Node N sends each node S_i a message consisting of N's identifier, N's IP address, and a multicast range (s_i, s_{i+1}) of the identifier space.[3] Node S_i will be responsible for multicasting the notification to nodes whose identifiers are in the range (s_i, s_{i+1}). Together, the j finger nodes of node N help N multicast its arrival to all nodes in the system.

Node S_i uses a similar process to expand the multicast tree with its own children. The purpose now is to cover nodes in range (s_i, s_{i+1}). Among nodes in its routing table, node S_i selects its finger nodes that are within range (s_i, s_{i+1}) as its children in the tree. Let P_j and p_i denote the children of node S_i and their identifiers, respectively. Node S_i asks node P_i to cover range (p_i, p_{i+1}), which in turn expands the tree by adding their finger nodes as children, and so forth. A node stops expanding the tree when it finds that there is no node in the multicast range it is assigned to.

The multicast "trees" in 1h-Calot are transient and purely conceptual. There is no message to construct the trees before use; no probing to maintain the trees; and no message to tear down the trees after use. Nodes expand the trees just in time based on local information. This allows successful multicast with inaccurate routing tables. Suppose node S is responsible for covering key range (a, b) and node P in that key range is missing from S's routing table. Node S will not select P as its child in the multicast tree even if P should be selected based on our definition of finger nodes. This mistake, however, will not prevent node P from receiving the notification. In the worst case, node P will receive the notification from its predecessor as the key range narrows.

When a node leaves, it notifies its predecessor and successor. The predecessor propagates this membership change to all nodes through a multicast tree rooted at itself, using a process similar to that for node arrivals. A node may fail without notice. Its predecessor detects this through lost heartbeats and then announces its departure. The average session duration in Gnutella is 2.9 hours [19], which is much shorter than the mean time to failure (MTTF) of most modern systems. We consider most node departures as voluntary rather than due to hardware or software failures. We recommend that the overlay software running on a node always notifies its predecessor when the user closes the application, which allows the predecessor to promptly multicast the node's departure thereby keeping the routing tables up to date.

3.2 Handling Failures

Without faults, each node receives a membership change notification through a multicast tree exactly once. Faults, however, are unavoidable. There are several scenarios in which a notification may not be propagated to some nodes. Suppose a node S in a multicast tree asks its child P to forward a notification to nodes in a key range that includes nodes $W_1, \cdots, W_j$. If P is no longer in the system, S will timeout due to the missing acknowledgment from P. S deletes P from its routing table, and tries using another node to forward the notification. The retrial may succeed but the notification has already been delayed such that some nodes hold inaccurate routing tables longer. If P dies after receiving and acknowledging the notification but before forwarding it, nodes $W_1, \cdots, W_j$ will miss this notification altogether.

Inaccurate routing tables do not persistently result in failed lookups so long as nodes properly maintain their predecessors and successors. However, inaccurate routing tables degrade routing performance by taking more hops (when live nodes are missing from the routing tables) or more frequently encountering failed hops (when dead nodes are kept in the routing tables). In a long-running environment, it is important to ensure that errors in the routing tables do not accumulate over time and eventually lead to an unacceptable routing performance. To this end, we propose node re-announcements to address the problem of missing live nodes, and routing entry timeouts to address the problem of stale dead nodes.

When node N joins, it multicasts a message to announce its arrival. Periodically, every h seconds afterwards, if node N is still alive, it multicasts a message to re-announce its existence. Nodes that missed previous announcements now have an opportunity to pick it up. Therefore, the number of missing nodes in a routing table does not accumulate over time. The period h is chosen such that the probability that a node lives longer than h seconds is $\frac{1}{2}$. Assuming node lifetime follows an exponential distribution $p_l(t)=\lambda_l e^{-\lambda_l t}$, where $\lambda_l=\frac{1}{l}$ and l is the average node lifetime, we have

$$\int_0^h p_l(t)dt = \frac{1}{2} \implies h = l \ln 2 \approx 0.7l. \tag{11}$$

Nodes need to know l in order to compute h. A node can locally estimate l by observing the lifetimes of nodes for which it received both birth and death notifications.

When a node P receives a notification regarding the existence of a node N, P adds N into its routing table and associates an h second timer with this routing entry. If node N is already in the routing table, node P resets the timer to h seconds. When the timer fires, node P deletes node N from its routing table. Ideally, if node N is always alive, node P receives N's re-announcements periodically and keeps N in the routing table. If node N dies and the notification fails to reach node P, P will purge N from its routing table after the timer fires. Therefore, the number of dead nodes in a routing table does not accumulate over time.

In summary, with timeouts and re-announcements, routing tables become soft-state images of the system. When the system stabilizes and no faults occur, the routing tables converge to a correct global view. The overhead, however, is the traffic for re-announcements as well as the cost for timer book-keeping. We quantify the traffic overhead below. A live node re-announces its existence every h seconds (see Equation 11) and half of the nodes leave before they make their first re-announcements. Hence, half of the nodes make their first re-announcements, among which half of them live long enough to make their second re-announcements, and so forth. The average number of re-announcements that a node makes during its lifetime is $\sum_{i=1}^{\infty}(\frac{1}{2})^i = 1$. During a node's lifetime, it multicasts a notification for its birth and death, respectively. Adding re-announcements increases multicast messages by 50%. The benefit is a soft-state protocol that handles faults cleanly.

3.3 Traffic Analysis

In this section, we compare the total traffic in 1h-Calot with that in traditional DHTs [20]. Simulation results in Section 5 show that 1h-Calot maintains accurate routing tables and resolves most lookups in one hop. Below we assume one-hop routing for all lookups. Each second, there are nf lookups in total. Lookup messages have size s and are acknowledged by packets of size $0.5s$. The traffic to process nf lookups is

$$L_o \approx (1 + 0.5)s \cdot nf. \tag{12}$$

Next, we calculate the traffic for maintenance. Each second, $\frac{n}{l}$ nodes join. We estimate that the notifications for node arrivals have size s (see Footnote 3). We assume that notifications are delivered to every node exactly once. The traffic to multicast notifications for node arrivals is

[3] In implementation, the message only needs to include node N's IP address and node S_{i+1}'s IP address since their identifiers are simply SHA-1 hashings of the IP addresses.

$$M_{o1} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (13)$$

The coefficient 0.5 is because messages are acknowledged by packets of size $0.5s$. On average, each node re-announces its existence once during its lifetime. The traffic for re-announcements is

$$M_{o2} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (14)$$

Each new node obtains a complete n-entry routing table from a node already in the overlay. A routing entry includes a node P's IP address and some properties such as P's bandwidth. It is not necessary to transmit P's identifier since the identifier is simply the SHA-1 hashing of the IP address. Copying a routing table is a bulk transfer; it does not incur per entry packet overhead or acknowledgment. We estimate the traffic to transmit one entry is $0.25s$ bytes. The traffic for copying routing tables is

$$M_{o3} = 0.25s \cdot \frac{n}{l} n. \quad (15)$$

Each second, $\frac{n}{l}$ nodes leave the system. The notification for a node departure contains only the IP address of the leaving node and a propagation range. We assume that the notification has size s. The traffic to propagate node departures is

$$M_{o4} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (16)$$

Every T=30 seconds, a node sends two heartbeats, one to its predecessor and one to its successor. We assume that the heartbeat messages have size $0.5s$. The traffic for heartbeats is

$$M_{o5} = 0.5s \cdot \frac{2n}{T}. \quad (17)$$

The total traffic (maintenance plus lookup) in 1h-Calot is

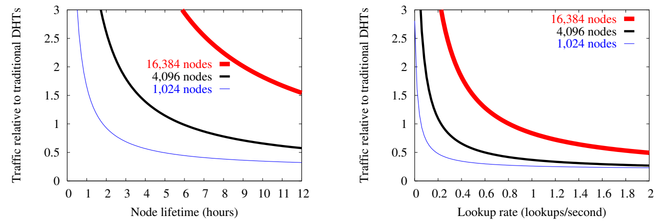
$$\begin{aligned} M_o &= L_o + M_{o1} + M_{o2} + M_{o3} + M_{o4} + M_{o5} \\ &= s \cdot n(1.5f + \frac{1}{T} + \frac{4.75n}{l}). \quad (18) \end{aligned}$$

Dividing M_o by M_{dht} in Equation 9, we get the relative traffic R_o between 1h-Calot and traditional DHTs:

$$R_o = \frac{M_o}{M_{dht}} \approx \frac{6.3}{fl} \cdot \frac{n}{\log_2 n}. \quad (19)$$

The traffic in 1h-Calot is dominated by the multicast traffic for routing table maintenance. For systems with up to a few thousand nodes, the maintenance traffic is well compensated for by the savings from efficient one-hop lookups. As a result, 1h-Calot can introduce less total traffic than traditional DHTs. The relative traffic R_o decreases as node lifetime l or lookup rate f increases. R_o grows with the system size (the $\frac{n}{\log_2 n}$ component), indicating that it is not economical to use 1h-Calot for very large systems. This problem is not unique to our design; it is inherent in any one-hop scheme [5, 7]. Membership update traffic in one-hop schemes grows quadratically with the system size, due to more frequent membership changes in a large system and the fact that each change is notified to more nodes.

Figure 4 plots the exact relative traffic R_o in Equation 19. When n=1,024 nodes, l=2.9 hours, and f=0.1 lookups/second, 1h-Calot saves traffic by 30%; when lookup rate f increases to 0.5, 1h-Calot saves traffic by 70%. In addition to the benefit of low traffic, 1h-Calot resolves lookups much faster than traditional DHTs, i.e., in one hop as opposed to $O(\log n)$ hops.



(a) f=0.1 lookups/node/second. (b) Node lifetime l=2.9 hours.

Figure 4: Relative traffic between 1h-Calot and traditional DHTs (the exact R_o in Equation 19, which grows with $O(\frac{1}{fl})$).

4. 2H-CALOT FOR TWO-HOP ROUTING

When the system is very large, efficient one-hop routing is no longer feasible. This is because the maintenance traffic in one-hop schemes grows quickly with $O(n^2)$ (see Equations 7 and 18). By contrast, the maintenance traffic in two-hop schemes that use $O(\sqrt{n})$ routing tables grows with $O(n^{1.5})$ (see Equation 8). When n is large, the difference between $O(n^2)$ and $O(n^{1.5})$ is significant, for instance, when n=10^6, $n^2/n^{1.5} = 1000$. Figure 2(b) shows that, even for very large systems (up to 20 million nodes), the total traffic of an "ideal" two-hop scheme can still be lower than that of traditional DHTs [20]. Moreover, two-hops schemes resolve lookups in two hops, much faster than traditional DHTs.

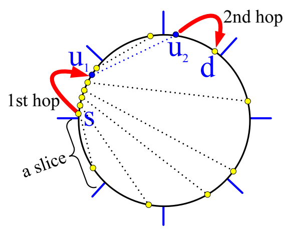
Our goal, therefore, is to design a practical two-hop protocol that approaches the performance of the "ideal" two-hop scheme. The main challenge is to maintain the large $O(\sqrt{n})$ routing tables in the face of frequent node joins and leaves and to do two-hop routing with the $O(\sqrt{n})$ routing tables in a peer-to-peer fashion. To this end, we propose our 2h-Calot protocol. Unlike existing hierarchical two-hop protocols [7], 2h-Calot is purely peer-to-peer. Below, we first present a "basic" version of 2h-Calot and then describe how to make it adaptive.

4.1 The Basic 2h-Calot

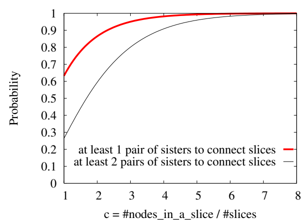
2h-Calot is a further development of 1h-Calot. It also organizes nodes into a ring topology. The "basic" version of 2h-Calot partitions the ring into continuous regions of equal size called *slices* (see Figure 5(a)), and runs a protocol similar to 1h-Calot inside each slice. A membership change that happens in a slice is only propagated to nodes in the same slice. Inside a slice, nodes know each other. 2h-Calot resolves a lookup in two hops. The first hop routes the lookup between the source slice and the destination slice. The second hop delivers the lookup within the destination slice.

Since nodes in the same slice know each other, the second hop is trivial. The challenge is to route between two arbitrary slices in one hop. For this purpose, each computer N runs two virtual nodes, N_0 and N_1, called *sister nodes*. N_0's identifier is the SHA-1 hashing of N's IP address and N_1's identifier is the SHA-1 hashing of N_0's identifier (i.e., double hashing of N's IP). Below we refer to "virtual nodes" simply as "nodes". To route a message between two slices, 2h-Calot tries to find a node in the source slice whose sister node sits in the destination slice to forward the message. That is, sister nodes act as gateways to connect different slices.

Suppose there are a large number of nodes S_i $(i = 1, \cdots, j)$ in a slice $\mathcal{S}$. The sister nodes P_i of nodes S_i are randomly distributed all over the identifier space because the node identifiers are generated randomly. Given an arbitrary destination slice $\mathcal{D}$, with a high probability, one of these sister nodes P_i may sit in slice $\mathcal{D}$. In other words, with a high probability, we can find a pair of sister nodes to connect slices $\mathcal{S}$ and $\mathcal{D}$.



(a) Illustration of 2h-Calot.



(b) Prob. of finding sister nodes to connect two random slices.

Figure 5: Highlights of the "basic" version of 2h-Calot.

Figure 5(a) is an illustration of 2h-Calot. Nodes at the two ends of a dashed link are sister nodes, e.g., nodes u_1 and u_2. Suppose node s in slice $\mathcal{S}$ wants to route a message to the node in slice $\mathcal{D}$ that is responsible for key d. Node s searches its routing table for a node u_1 in the local slice $\mathcal{S}$ whose sister node u_2 resides in the destination slice $\mathcal{D}$. Node s sends the message to node u_1. Nodes u_1 and u_2 are two virtual nodes running on the same computer. Node u_2 then directly forwards the message to the destination since node u_2 knows all nodes in slice $\mathcal{D}$.

We next derive a proper configuration for 2h-Calot. We want the slices to be small such that the traffic for membership updates inside slices is low. But we also want the slices to be sufficiently large such that the probability of finding two sister nodes to connect two random slices is high. Let k denote the number of slices, m denote the number of nodes in a slice, and n denote the number of computers. $k \cdot m = 2n$ since each computer runs two virtual nodes. Let $c = \frac{m}{k}$ be the main parameter for 2h-Calot. We have

$$\text{number of slices:} \quad k = \sqrt{2n/c} \tag{20}$$

$$\text{number of nodes in a slice:} \quad m = \sqrt{2cn}. \tag{21}$$

Let $\mathcal{S}$ and $\mathcal{D}$ denote two random slices. There are k slices in total. The sister of a node is randomly distributed in the identifier space. For a node in slice $\mathcal{S}$, the probability that its sister node is in slice $\mathcal{D}$ is $p = \frac{1}{k}$. Among the m nodes in slice $\mathcal{S}$, on average $c = m/k$ nodes have sister nodes in slice $\mathcal{D}$. The probability that exactly x nodes in slice $\mathcal{S}$ have sister nodes in slice $\mathcal{D}$ follows a Binomial distribution:

$$P(X{=}x) = \binom{m}{x} p^x (1-p)^{m-x} \approx e^{-c} \cdot \frac{c^x}{x!} \tag{22}$$

$$P(X{\geq}1) = 1 - P(X{=}0) \approx 1 - e^{-c} \tag{23}$$

$$P(X{\geq}2) = 1 - P(X{=}0) - P(X{=}1) \approx 1 - e^{-c} - ce^{-c}. \tag{24}$$

The approximation above exploits the fact that this Binomial distribution approaches a Poisson distribution when m is large.

$P(X \geq 1)$ is the probability that there exists at least one pair of sister nodes to connect two random slices; $P(X \geq 2)$ is the probability that there exist at least two pairs of sister nodes to connect two random slices. Figure 5(b) plots $P(X \geq 1)$ and $P(X \geq 2)$. This figure shows that, with a high probability, we can find sister nodes to connect two random slices. Hence the two-hop routing in Figure 5(a) can be accomplished. We opt for configuration $c = 5$.

$$c = 5 \implies m = \sqrt{10n}, \quad k = \sqrt{0.4n}, \tag{25}$$

$$P(X \geq 1) \approx 0.993, \quad P(X \geq 2) \approx 0.960 \tag{26}$$

With this configuration, the probability of finding more than one pair of sister nodes to connect two slices is also high (0.960). This offers an opportunity to consider network proximity when routing messages among slices. If there exists more than one node to reach the destination slice, we can choose the node that has the lowest latency to forward the message. Currently, our simulator does not exploit proximity-aware routing.

4.2 Making 2h-Calot Adaptive

Ideally, the number of nodes in a slice ($m = \sqrt{2cn}$) and the number of slices ($k = \sqrt{2n/c}$) should automatically adapt as the system size n changes. In existing solutions for two-hop routing [7, 8], nodes need to unanimously agree upon the number of slices, making it impossible to do decentralized adaptations based on only local knowledge. Below, we show how to make 2h-Calot adaptive.

The key observation is that, the use of "slices" in 2h-Calot is completely artificial. So long as a node knows a sufficient number of nodes randomly distributed in the identifier space, given a message to any destination, it can route the message in one hop to a place very close to the destination by using one of those random nodes (*the first hop*). Furthermore, so long as each node knows a sufficient number of neighbors along the ring, the message can be delivered to its destination in one hop when it is already at a place very close to the destination (*the second hop*). Hence, 2h-Calot is able to accomplish two-hop routing without using "slices".

More specifically, the routing table of a node N includes its $\frac{m}{2}$ clockwise neighbors along the ring and $\frac{m}{2}$ counter-clockwise neighbors along the ring. We refer to the continuous range in the identifier space that spans over these m neighbors as node N's *neighbor zone*. Neighbor zones essentially replace the role of slices in Figure 5(a). Unlike the fixed slices, each node has its own neighbor zone centered at itself and need not know the neighbor zones of others. Below we always assume that, whenever a node N knows about a node P, N automatically knows about P's sister. Hence there are actually $2m$ nodes in a node's routing table: m nodes in its neighbor zone (the "neighbor set") and their m sisters (the "sister set") that are randomly scattered in the identifier space.

The routing algorithm is the same as that in 1h-Calot. Given a lookup, a node N greedily forwards the lookup to the node P that is, to N's knowledge, the closest in absolute distance to the lookup key. Node P either returns the object or further forwards the lookup greedily. When a node N searches its routing table for a node P that is closest to the destination, N does not distinguish between whether P is from its "neighbor set" or its "sister set". Nodes have no notion of "slices" either. The only rule is greedy forwarding. As in 1h-Calot, correct routing is guaranteed so long as each node correctly maintains its predecessor and successor.

4.3 Routing Table Maintenance

The maintenance protocol for 2h-Calot is similar to that for 1h-Calot but with a major difference: when a node joins or leaves, the multicast notification is only sent to its $\frac{m}{2}$ clockwise neighbors and $\frac{m}{2}$ counter-clockwise neighbors along the ring, rather than all nodes in the system. Nodes do not know the exact number n of computers in the system. They estimate n and m from local knowledge. The processes for announcing node arrivals and departures are similar. Below we use a node arrival as the example.

When a new computer N joins, it functions as two virtual nodes N_j (j=0, 1). N_0's identifier is the SHA-1 hashing of N's IP address and N_1's identifier is the SHA-1 hashing of N_0's identifier (i.e., double hashing of N's IP). Nodes N_0 and N_1 execute the same protocol but function independently as if they were "real" nodes. Below we use N_j to refer to either of them. Node N_j joins the ring topology and obtains a copy of the routing table from its predecessor P. Suppose the routing table includes a total of y neighbors of P, either clockwise or counter-clockwise. The neighbors of node

P are also neighbors of node N_j. Node N_j adds into its routing table the y neighbors and node P. Suppose the size of the continuous region of the identifier space spanned over by the $y+2$ neighbors (including nodes P and N_j) is z. N_j estimates the total number of computers in the system as

$$\text{estimated total computers:} \quad n = \frac{1}{2}\frac{2^{160}}{z}(y+2). \tag{27}$$

The size of N_j's neighbor zone is estimated as $b = \frac{2^{160}}{k}$, where $k = \sqrt{2n/c}$. Suppose N_j's identifier is d. N_j's neighbor zone is

$$\text{estimated neighbor zone:} \quad \mathcal{K} = [\, d - \tfrac{1}{2}b, d + \tfrac{1}{2}b\,]. \tag{28}$$

Note that the operations are in modulo 2^{160}. Node N_j purges from its routing table neighbors that are outside $\mathcal{K}$. Different nodes may estimate the sizes of their neighbor zones differently. Since the "slices" (neighbor zones) are configured to be sufficiently large ($c = \frac{m}{k} = 5$), the variance of the estimation is well tolerated. With a high probability, a node can forward a message in one hop to any region in the identifier space through the sisters of nodes in its neighbor zones.

Node N_j needs to multicast a notification about its arrival to all nodes in its neighbor zone $\mathcal{K}$. The multicast process is similar to that of 1h-Calot but the notification is propagated both clockwise and counter-clockwise. In 1h-Calot, the finger nodes of a node are defined as the successor nodes of keys $r_i = k + 2^i$ $(i = 0, ..., 159)$. In 2h-Calot, the *forward-finger* nodes of a node are defined as the successor nodes of keys $r_i = k + 2^i$ $(i = 0, ..., 158)$ and the *backward-finger* nodes are defined as the predecessor nodes of keys $r_i = k - 2^i$ $(i = 0, ..., 158)$. In the identifier space, the finger nodes of a node distribute at exponentially increasing distance from the node, either clockwise or counter-clockwise.

The multicast process to cover nodes in node N's neighbor zone $\mathcal{K} = [d - \frac{1}{2}b, d + \frac{1}{2}b]$ works as follows. Node N splits $\mathcal{K}$ into a backward range $\mathcal{K}_b = [d - \frac{1}{2}b, d]$ and a forward range $\mathcal{K}_f = [d, d + \frac{1}{2}b]$. It multicasts notifications through two different trees T_b and T_f to cover ranges $\mathcal{K}_b$ and $\mathcal{K}_f$ separately. The tree T_f is constructed using the links between nodes and their forward-finger nodes. The multicast process over tree T_f is exactly the same as that in 1h-Calot (see Figure 3). The multicast process in tree T_b is the same as that in tree T_f except that the notification travels over links between nodes and their backward-finger nodes.

Like 1h-Calot, 2h-Calot also uses timeouts and re-announcements to make the routing tables soft-state images of the system. Before a re-announcement, a node always re-estimates the system size n and its neighbor zone $\mathcal{K}$. The re-announcement will cover nodes in the updated neighbor zone. This helps nodes with a long lifetime adapt as the system evolves.

4.4 2h-Calot vs. Other Two-hop Schemes

Existing protocols for two-hop routing also partition the overlay into slices and nodes in the same slice know each other [7, 8]. There are several major differences between 2h-Calot and these hierarchical protocols. **(1)** 2h-Calot is purely peer-to-peer and extremely simple. Each node knows $O(\sqrt{n})$ neighbors along the ring—*That's it!* Notification multicast uses these neighbors; routing also uses these neighbors. There are neither "slices" nor multicast "trees" to maintain; both are conceptual. By contrast, existing hierarchical protocol [7] partitions the overlay into "units" and "slices", and designates nodes as "slice leaders", "unit leaders", "ordinary nodes", and "slice representatives". Nodes have different roles and run different protocols. **(2)** 2h-Calot distributes load evenly across nodes. Each pair of sister nodes carry some traffic between two "slices". By contrast, for each slice, existing protocols select a few nodes to act as gateways to carry all incoming traffic from other slices. **(3)** 2h-Calot estimates neighbor zones from local knowledge and adapts as the system evolves. By contrast, existing protocols use fixed slices and cannot adapt easily.

4.5 Traffic Analysis

We compare the traffic in 2h-Calot with that in traditional DHTs [20] through analysis. The process is similar to that for 1h-Calot, but there are $2n$ virtual nodes for a system with n computers and each notification is sent to only $m = \sqrt{2cn} = \sqrt{10n}$ nodes.

Simulation results in Section 5 show that 2h-Calot maintains very accurate routing tables and resolves most lookups in two hops. As an approximation, we assume two-hop routing for all lookups. Each second, there are nf lookups in total. Lookup messages have size s and are acknowledged by packets of size $0.5s$. The traffic to process nf lookups is

$$L_t \approx (1 + 0.5)s \cdot 2nf. \tag{29}$$

Next, we calculate the traffic for maintenance. Each second, $\frac{2n}{l}$ new nodes (or $\frac{n}{l}$ computers) join. Conceptually, the notification for a node join includes its IP address, its identifier, its sister node's identifier, and a boundary and direction (clockwise or counter-clockwise) for the notification to be propagated. We estimate the notification message has size s. (Note that the identifiers and boundaries are simply SHA-1 hashings of the IP addresses and we need not transmit them. See Footnote 3). We assume that each notification is delivered to $m = \sqrt{10n}$ nodes exactly once. The traffic to multicast notifications for node arrivals is

$$M_{t1} = (1 + 0.5)s \cdot \frac{2n}{l} m. \tag{30}$$

The coefficient 0.5 is because messages are acknowledged by packets of size $0.5s$. On average each node re-announces its existence once during its lifetime. The traffic for re-announcements is

$$M_{t2} = (1 + 0.5)s \cdot \frac{2n}{l} m. \tag{31}$$

Each new node copies a routing table from its predecessor. Conceptually, the routing table includes m neighbors and the sisters of those m neighbors. We only need to transfer the IP addresses of the m computers since the $2m$ identifiers are just SHA-1 hashings of the IP addresses. Copying a routing table is a bulk transfer; it does not incur per entry packet overhead or acknowledgment. We estimate the traffic to transmit one entry of the routing is $0.25s$. The traffic for copying routing tables is

$$M_{t3} = 0.25s \cdot \frac{2n}{l} m \tag{32}$$

Each second, $\frac{2n}{l}$ nodes leave the system. The notification for a node departure contains only the IP address of the leaving node and a propagation range. We assume that the notifications have size s. The traffic to propagate node departures is

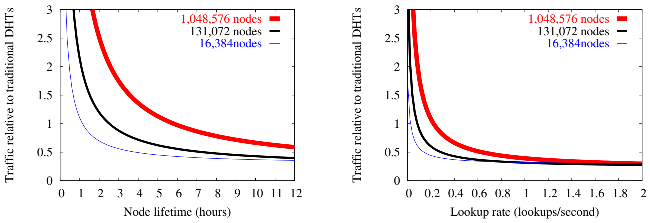
$$M_{t4} = (1 + 0.5)s \cdot \frac{2n}{l} m. \tag{33}$$

Every T=30 seconds, a node sends two heartbeats, one to its predecessor and one to its successor. We assume that the heartbeat messages have size $0.5s$. There are $2n$ nodes in total. The traffic for heartbeats is

$$M_{t5} = 0.5s \cdot \frac{4n}{T}. \tag{34}$$

The total traffic (maintenance plus lookup) for 2h-Calot is

$$\begin{aligned} M_t &= L_t + M_{t1} + M_{t2} + M_{t3} + M_{t4} + M_{t5} \\ &= s \cdot n(3f + \frac{2}{T} + \frac{9.5\sqrt{10n}}{l}). \end{aligned} \tag{35}$$



(a) (b)

Figure 6: Relative traffic between 2h-Calot and traditional DHTs (the exact R_t in Equation 36, which grows with $O(\frac{1}{fl})$). (a) Vary node lifetime (lookup rate f=0.1). (b) Vary lookup rate (node lifetime l=2.9 hours).

Comparing Equations 18 and 35, we see that 2h-Calot is more scalable than 1h-Calot. 2h-Calot's traffic grows with $O(n^{1.5})$ while 1h-Calot's traffic grows with $O(n^2)$.

Dividing M_t by M_{dht} in Equation 9, we get the relative traffic R_t between 2h-Calot and traditional DHTs:

$$R_t = \frac{M_t}{M_{dht}} \approx \frac{40}{fl} \cdot \frac{\sqrt{n}}{\log_2 n}. \quad (36)$$

Like 1h-Calot, the traffic for 2h-Calot is dominated by membership updates. The maintenance traffic, however, is well compensated for by the savings from efficient two-hop lookups when $fl \geq 1000$.

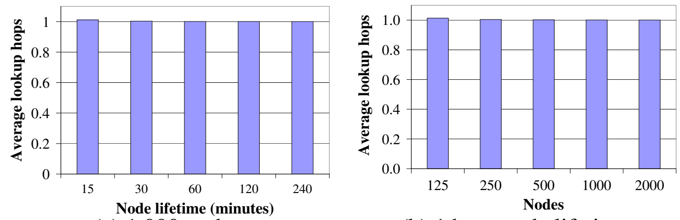
Figure 6 plots the exact R_t in Equation 36. When n=131,072 computers and f=0.1 lookups/second, 2h-Calot saves traffic by 10%; when lookup rate f increases to 0.5, 2h-Calot saves traffic by 61%. When the lookup rate f further increases to 1, 2h-Calot saves traffic by 61% even for a 1,048,576-node system. In addition to the benefit of low traffic, 2h-Calot resolves lookups much faster than traditional DHTs, i.e., in two hops as opposed to $O(\log n)$ hops.

5. EXPERIMENTAL RESULTS

We built an event-driven simulator to evaluate 1h-Calot and 2h-Calot. The simulator consists of 5,500 lines of C++ code. It simulates a complete system, including dynamic node arrivals and departures, timeouts, and network delays. We do not simulate the network-level packet details. Limited by the 2GB memory of our computers, we can simulate 1h-Calot with up to 2,000 nodes and 2h-Calot with up to 16,000 computers (i.e., 32,000 virtual nodes).

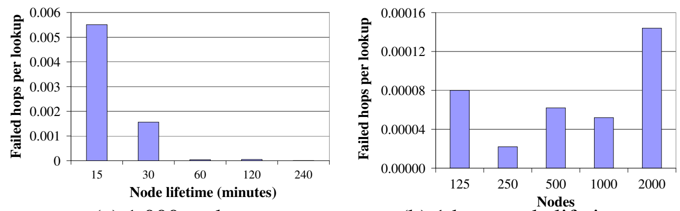
Modeling network topologies and latencies is still an open research topic. We follow the approach [6] that focuses on ensuring the simulated network latencies follow the distribution of real network latencies in the Internet. In our simulator, the network latencies between nodes are randomly sampled from the King dataset [3], which is extracted from real measurements of the round-trip times (RTTs) between 2,048 DNS servers. We divide the RTTs by two to obtain one-way latencies. Excluding the empty entries in the RTT matrix, the average one-way latency is 91ms.

Unless otherwise noted, the simulation works as follows. The system starts with one node and continuously adds more nodes until the population reaches n. From then on, node arrival is a Poisson process with rate $\lambda_e = \frac{n}{l}$. Node lifetime follows an exponential distribution with a mean l. The system population stabilizes around n as nodes join and leave. After the system undergoes $10n$ membership changes, i.e., a total of $10n$ nodes have joined or left the system, the simulator enters the evaluation phase. It takes a snapshot of the routing tables and uses them to evaluate routing performance, during which each node on average issues 1,000 random lookups. Our simulator models the lookups that a node issues as a Poisson process with rate f. However, unless related statistics are needed, the simulator does not fully execute the lookups issued before the evaluation phase. We found this optimization important to make the simulation time manageable when the system size is large. This optimization makes the reported lookup performance more pessimistic because the overlooked lookups can help detect and fix some inaccurate entries in the routing tables.



(a) 1,000 nodes. (b) 1 hour node lifetime.

Figure 7: Routing hops per lookup in 1h-Calot.



(a) 1,000 nodes. (b) 1 hour node lifetime.

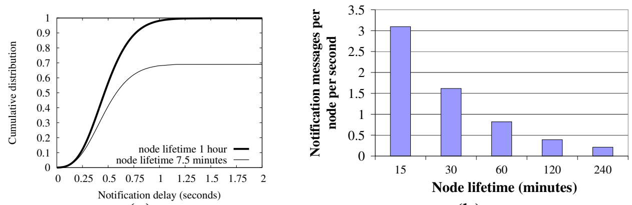
Figure 8: Failed routing hops per lookup in 1h-Calot.

Below, we present results regarding various aspects of 1h-Calot and 2h-Calot, including traffic, routing performance, resilience in the face of membership changes, and the ability to adapt as the system size evolves.

5.1 1h-Calot

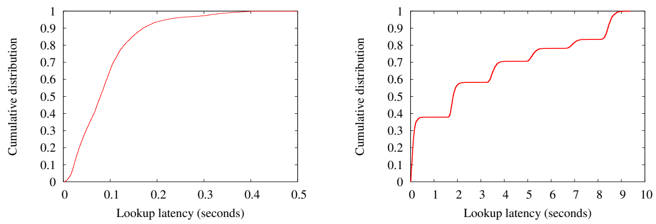
We first present results on 1h-Calot. Figures 7(a) and 7(b) show the average routing hops per lookup when varying node lifetime and system size, respectively. In both figures, the lookup hops are very close to one, indicating that the routing tables are very accurate. For instance, with a one hour lifetime and one thousand nodes, the average routing hops are only 1.0008. In Figure 7(a), the routing performance improves as the node lifetime increases. The absolute improvement, however, is small because the routing hops are already very close to one.

Figure 8 reports the average number of failed hops encountered per lookup. (Note that a failed hop does not necessarily lead to an irresolvable lookup. The system always retries alternative routing paths.) Both missing live nodes and listed dead nodes can lead to inaccurate routing tables, among which the latter is particularly harmful as it significantly increases lookup latencies. In our simulator, it takes a timeout that is 18 times of the average one-way network latency to detect a failed hop before trying an alternative. Figure 8(a) shows that the failed hops w reduce dramatically as the node lifetime increases. With a half an hour lifetime, w=0.0016; with a one hour lifetime, w=0.000052 (1 failed hop out of 20,000 lookups). Comparing Figures 8(a) and 8(b), we see that the failed hops are much more sensitive to node lifetime than to system size. Comparing Figures 7(a) and 8(a), we find that the number of failed hops is a more revealing metric of 1h-Calot's performance than the number of lookup hops because of the high cost of failed hops.



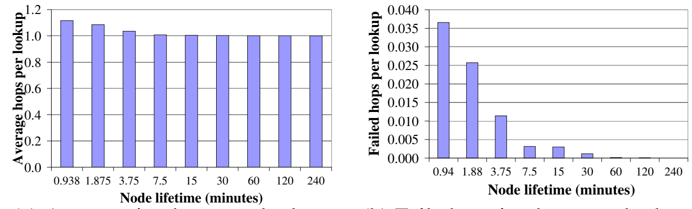
(a) (b)

Figure 9: A 1,000-node 1h-Calot. (a) Delivery delay of multicast notifications. (b) The number of notifications that a node receives per second.



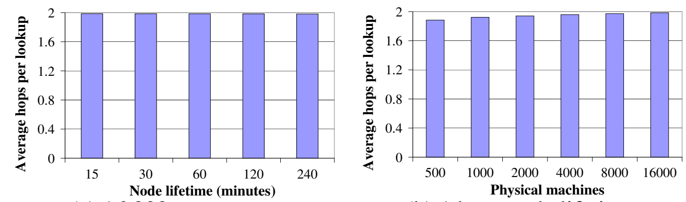
(a) 1 hour node lifetime. (b) 7.5 minute node lifetime.

Figure 10: CDF of lookup latency for a 1,000-node 1h-Calot.



(a) Avg. routing hops per lookup. (b) Failed routing hops per lookup.

Figure 11: The routing performance of a 1,000-node 1h-Calot that uses "redundant flooding" to handle churn.



(a) 16,000 computers. (b) 1 hour node lifetime.

Figure 12: Average routing hops per lookup in 2h-Calot.

In Figure 8(b), the number of failed hops fluctuates—it is not monotonic with respect to system size. The system size has several conflicting impacts on the accuracy of the routing tables. As the system becomes larger, more nodes join and leave per second. Therefore, there are more notifications and nodes communicate with their finger nodes more frequently, which helps nodes detect dead finger nodes faster. As a result, notifications may propagate more reliably. On the other hand, as the system grows, the height of the multicast trees increases, which delays notifications and increases the chance of encountering dead nodes during a multicast. Because of these conflicting factors, the number of failed hops during a lookup is not monotonic with respect to system size. The results from our approximate analysis match the trend of the simulation results. The analysis is omitted due to space limitation.

Figure 9(a) plots the cumulative distribution of the time that it takes to deliver a membership change notification from the source to other nodes. When the node lifetime is one hour, 98% of the nodes receive the notification within one second after the membership change occurs. This quick and reliable distribution of membership changes is the key reason why 1h-Calot can maintain accurate routing tables. When the node lifetime reduces to 7.5 minutes, the delay of notifications is significantly longer, due to disruptions in the multicast process caused by dead nodes. Only about 70% of the nodes receive the notification within 2 seconds (we dot not show nodes that receive the notification after 2 seconds). Figure 9(b) reports the average number of notification messages a node receives per second. With a 2 hour node lifetime (2.9 hours in Gnutella [19]), a node on average receives 0.39 notifications per second. In 1h-Calot, all notifications that a node forwards go through the $O(\log n)$ links to its finger nodes. When the message rate is high, the node can potentially aggregate notifications for the same outgoing link and send them in a single packet. Our simulator currently does not implement this feature.

Figures 7 and 8 show that the average routing performance is good. In Figure 10, we plot the cumulative distribution of lookup latencies in a 1,000-node system. In Figure 10(a), the latency for a very small fraction of nodes is longer than 0.5 seconds. We do not plot them in order to make the figure readable. With a one hour node lifetime, the 95th percentile lookup latency is only 220ms. For extremely short node lifetimes (7.5 minutes), the lookup latency is much higher. The saw-like curve is due to the timeout and retrial that can occur multiple times during a lookup. The results in Figure 10(b) and Figure 9 suggest that the basic 1h-Calot is not suitable for environments with a high churn rate.

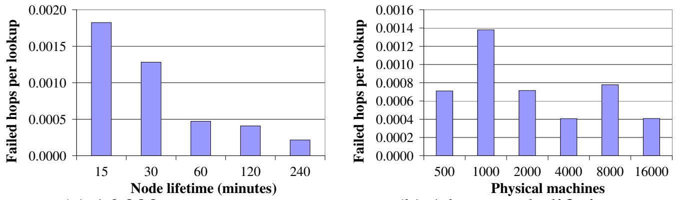
When the node lifetime is extremely short, one way to improve the reliability of the dissemination of membership changes is to propagate notifications through redundant paths rather than through a single tree, for instance, by flooding a notification through each of the $O(n \log n)$ links between nodes and their finger nodes. When a node receives a notification, it forwards the notification to each of its $O(\log n)$ finger nodes. Each node may receive a notification up to $O(\log n)$ times from different incoming links. We call this method "redundant flooding". It improves reliability at the expense of increased traffic. Figure 11 shows that, with this method, 1h-Calot can achieve good routing performance even under high churns.

5.2 2h-Calot

1h-Calot and 2h-Calot share many features. Our evaluation of 2h-Calot will focus on the aspects unique to 2h-Calot.

Figure 12 plots the average routing hops in 2h-Calot, which are very close to two, even slightly under two when the system size is small. This is because 2h-Calot sometimes resolves lookups in one hop, when the destination happens to sit in the query initiator's neighbor zone and when the destination happens to be the sister node of a node in the query initiator's neighbor zone. Compared with 1h-Calot, 2h-Calot's performance is closer to the ideal case and less sensitive to node lifetime. This is because a node's neighbor zone contains a medium number of nodes, e.g., 400 nodes for the configuration in Figure 12(a). Furthermore, 2h-Calot uses both forward-finger nodes and backward-finger nodes to disseminate a notification through two disjoint trees, which is faster than using just one tree in 1h-Calot. For Figure 12(a), the number of nodes in one tree is 200, equivalent to a small 1h-Calot system.

Figure 13 plots the failed routing hops per lookup. Like 1h-Calot, 2h-Calot maintains very accurate routing tables and the failed



(a) 16,000 computers. (b) 1 hour node lifetime.

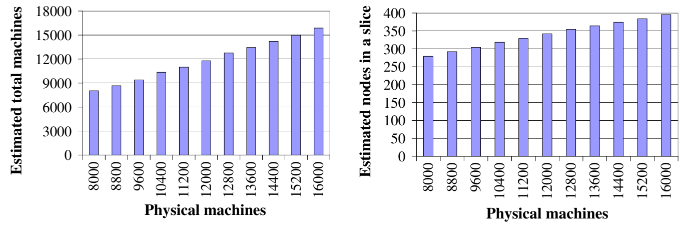
Figure 13: Failed routing hops per lookup in 2h-Calot.

hops are very low. With 16,000 computers and a one hour node lifetime, it encounters only one failed hop out of every 2,000 lookups. For similar reasons to that for 1h-Calot, the failed hops in Figure 13(b) fluctuate as the population grows. Comparing Figures 13 and 8, we see that the failed hops per lookup in 2h-Calot is higher than that in 1h-Calot. This is because 2h-Calot resolves lookups in two hops and the chance of encountering dead nodes is higher.

In Figures 14 and 15, we evaluate 2h-Calot's ability to adapt as the system size doubles over a short period of time. The simulation works as follows. The system starts with one computer and continuously adds more computers until the population reaches n=8,000 computers (16,000 virtual nodes). From then on, computer arrival is a Poisson process with rate $\lambda_e = \frac{n}{l}$. Node lifetime follows an exponential distribution with a mean l=1 hour. The population stabilizes around n until a total of $10n$ computers have joined or left. The system then enters the second phase to grow the population. The computer arrival rate is increased by 10% to $\lambda_e = (1 + \frac{1}{10})\frac{n}{l}$. The average population then starts to grow although population fluctuations still exist due to the randomness. The arrival rate λ_e stays at that level until the population reaches $(1+\frac{1}{10})n$ for the first time. Then the arrival rate is increased again to $\lambda_e = (1+\frac{2}{10})\frac{n}{l}$ and stays at that level until the population reaches $(1+\frac{2}{10})n$. Generally, the arrival rate stays at level $\lambda_e = (1+\frac{i}{10})\frac{n}{l}$ $(i = 1, \cdots, 10)$, until the population reaches $(1+\frac{i}{10})n$. The simulation ends when the population reaches $2n$. In 21 simulated hours, the number of computers doubles from 8,000 to 16,000. This fast growth is a stress test for 2h-Calot's adaptation ability.

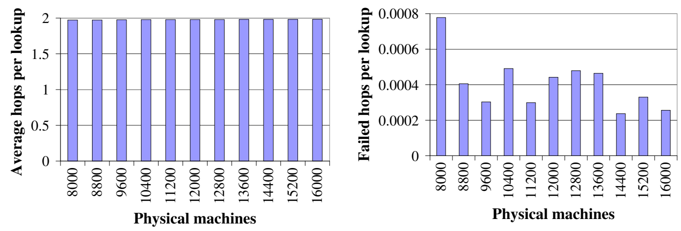
In 2h-Calot, nodes use Equation 27 to estimate the total number of computers and use Equation 28 to estimate their neighbor zones. Figure 14 plots estimated total computers and the number of computers that a node keeps in its routing table (i.e., computers in a node's neighbor zones). Both are averaged over all nodes and are presented as functions of the growing system size. Although the estimations are derived from local knowledge, they are very accurate and adapt automatically as the system grows. In 2h-Calot, nodes need not have a consistent view about the "slices". Each node has its own neighbor zone centered at itself and the sizes of the neighbor zones are updated locally and dynamically without affecting others. This is the key reason why 2h-Calot can adapt while other two-hop schemes cannot [7, 15, 18]. Figure 15 plots the average number of routing hops and failed hops as the system grows. The average hops are close to two and the failed hops are extremely low. These results are similar to the previous results when the computer arrival rate is constant, indicating that evolving system size is not a major adverse factor for 2h-Calot, owing to its ability to adapt.

Lastly, we evaluate 2h-Calot's sensitivity to its only major parameter c, which decides the number of nodes in a neighbor zone ($m = \sqrt{2cn}$). Table 2 shows the average routing hops per lookup as a function of the parameter c. Consistent with the analysis in Equation 22, the probability of resolving lookups in two hops is



(a) Estimated population (Eq 27). (b) Nodes in est. zones (Eq 28).

Figure 14: 2h-Calot's adaptation ability as the population grows from 8,000 to 16,000 computers (one hour node lifetime).



(a) Routing hops per lookup. (b) Failed hops per lookup.

Figure 15: 2h-Calot's adaptation ability as the population grows from 8,000 to 16,000 computers (one hour node lifetime).

high when $c \geq 3$. Larger c leads to higher traffic because each zone is larger and each membership change is sent to more nodes. We choose $c = 5$, which is sufficient to guarantee two-hop routing with a high probability and also provides a buffer so that dynamic changes in population can be handled effectively.

$c = m/k$	1	2	3	4	5	6	7
avg. hops	2.431	2.131	2.035	1.999	1.984	1.976	1.972

Table 2: Routing hops per lookup while varying the parameter c (16,000 computers, one hour node lifetime).

6. RELATED WORK

Recent works have extensively compared DHTs with routing tables of size $O(\log n)$ [6, 12, 13, 23]. We instead introduce total traffic as the uniform metric to compare DHTs with $O(1)$ to $O(n)$ routing tables. Xu et al. [23] studied the tradeoff between routing table size and network diameter. Several degree-diameter optimal DHTs have been proposed [9, 11, 13, 14]. We intend to answer the question of "given so many degree-diameter optimal DHTs, what is the routing table size that minimizes the total traffic and how to implement it in a practical, peer-to-peer fashion?"

The most relevant work in one-hop and two-hop routing is done by Gupta et al. [7] In their one-hop scheme, membership changes are propagated through a single pre-determined hierarchy. Unlike our peer-to-peer 1h-Calot protocol, nodes in this scheme have different roles and run different protocols. Slice leaders have a much higher load than others and the system critically relies on them to function. They recommended this scheme for systems with up to a few million nodes. Under their recommended configuration, a slice leader must keep track of and directly send notifications to 5,000 other slice leaders. They also proposed a hierarchical two-hop scheme. See Section 4.4 for a detailed comparison between this two-hop scheme and our 2h-Calot.

Beehive [16] replicates objects according to object popularities to achieve $O(1)$ lookups. There are applications in which the short lifetimes of objects make them unsuitable for replication; and there are applications that have no objects to replicate at all, for instance, message indirection. The fundamental functionality of DHTs is routing. Calot addresses this fundamental problem and has wider applications than Beehive.

Like 2h-Calot, Kelips [8] also maintains $O(\sqrt{n})$ routing tables to achieve $O(1)$ routing. Kelips uses gossips to disseminate membership changes. Gossips are not efficient in traffic because a node may receive the same notification multiple times. More importantly, the gossip protocol takes time $O(\sqrt{n}\log^3(n))$ to propagate a membership change throughout the entire system—over an hour for systems with 10^5 or 10^6 nodes.

Mizrak et al. [15] proposed a hierarchical two-hop system, in which all incoming traffic to a slice goes through the slice leader. HiScamp [5] is a hierarchical protocol that uses gossips to propagate membership information. Rodrigues et al. [18] proposed a one-hop scheme that uses well-provisioned special servers to inform other nodes of the system configuration.

7. CONCLUSIONS

In this paper, we compared DHTs with $O(1)$ to $O(n)$ routing tables and proposed practical traffic-reducing DHT designs that use large routing tables. We made the following contributions.

- We modeled and analyzed the traffic in DHTs, taking into account both maintenance cost and lookup cost. Our analysis suggests that the most traffic-efficient routing table size grows with $O(fl\ln(n))$, where f is the lookup rate, l is the node lifetime, and n is the number of nodes. For realistic systems like Open DHT [22], we assume a node on average processes 1,000 or more lookups during its lifetime, i.e., $fl \geq 1000$. Under this assumption, our analysis shows that large routing tables lead to both fast lookups and low traffic.

- We proposed 1h-Calot, a purely peer-to-peer one-hop protocol, which is efficient for systems with up to a few thousand nodes. By contrast, existing one-hop protocols are hierarchical. 1h-Calot maintains $O(n)$ routing tables by multicasting node arrivals and departures through n different trees.

- We proposed 2h-Calot, a purely peer-to-peer and adaptive two-hop protocol, which is efficient for systems with up to a few million nodes. By contrast, existing two-hop protocols are hierarchical and cannot adapt. In 2h-Calot, each computer runs two virtual sister nodes with random identifiers. Each node knows $O(\sqrt{n})$ neighbors along the ring and the sisters of these neighbors. A node uses this information for both routing and membership change multicast.

Both 1h-Calot and 2h-Calot are extremely simple: multicast maintains the routing tables; information in the routing tables is then used to guide multicast and routing. Compared with traditional DHTs that use $O(\log n)$ routing tables, 1h-Calot and 2h-Calot save total traffic by up to 70% under typical workloads, while resolving lookups in one or two hops as opposed to $O(\log n)$ hops. However, we acknowledge that 1h-Calot and 2h-Calot are not designed for environments with high churns, e.g., several minute node lifetimes.

The optimal routing table size is proportional to $O(fl\ln(n))$. Currently, 1h-Calot and 2h-Calot use $O(n)$ and $O(\sqrt{n})$ routing tables for certain typical workloads. An ideal design should adapt as f, l, and n change. This is an interesting subject of future research. In addition, our "redundant flooding" method uses $\log_2(n)$ times redundant traffic to improve the reliability of membership change notification. We are working on methods that allow us to control the degree of redundancy according to the stability of the system.

Acknowledgments

We thank Gautam Altekar for his contributions to this project. We thank Chun Zhang, the anonymous reviewers, and our shepherd for their valuable feedback. Work at the University of Rochester was supported by NSF grants CCR-0219848, ECS-0225413, CNS-0411127, CCR-9988361, and EIA-0080124; by the U.S. Department of Energy Office of Inertial Confinement Fusion under Cooperative Agreement No. DE-FC03-92SF19460; and by a Faculty Partnership Award from IBM.

REFERENCES

[1] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *HotOS*, 2003.

[2] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP*, 2001.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a DHT for Low Latency and High Throughput. In *NSDI*, 2004. The network latency data set is available at http://www.pdos.lcs.mit.edu/p2psim/kingdata.

[4] M. J. Freedman, E. Freudenthal, and D. Maziéres. Democratizing Content Publication with Coral. In *NSDI*, 2004.

[5] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. HiScamp: self-organising hierarchical membership protocol. In *European ACM SIGOPS Workshop*, 2002.

[6] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM*, 2003.

[7] A. Gupta, B. Liskov, and R. Rodrigues. Efficient Routing for Peer-to-Peer Overlays. In *NSDI*, 2004.

[8] I. Gupta, K. Birman, P. Linga, A. Demers, and R. V. Renesse. Kelips: building an efficient and stable P2P DHT through increased memory and background overhead. In *IPTPS*, 2003.

[9] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *IPTPS*, 2003.

[10] KaZaA. http://www.kazaa.com.

[11] S. Kumar, A.and Merugu, J. Xu, and X. Yu. Ulysses: A Robust, Low-Diameter, Low-Latency Peer-to-peer Network. In *ICNP*, 2003.

[12] J. Li, J. Stribling, T. Gil, R. Morris, and F. Kaashoek. Comparing the performance of distributed hash tables under churn. In *IPTPS*, 2004.

[13] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM*, 2003.

[14] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *PODC'02*, 2002.

[15] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In *WIAPP*, 2003.

[16] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *NSDI*, 2004.

[17] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *USENIX Annual Technical Conference*, 2004.

[18] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system. In *SIGOPS European Workshop*, 2002.

[19] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, 2002.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[21] C. Tang and S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *NSDI*, 2004.

[22] The Open DHT Project. http://openhash.org/.

[23] J. Xu, A. Kumar, and X. Yu. On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks. *JSAC*, 22(1):151–163, January 2004.