RC23515 (W0502-041) February 4, 2005
Computer Science

IBM Research Report

On the Tradeoff among Capacity, Routing Hops, and Being Peer-to-Peer in the Design of Structured Overlay Networks

Chunqiang Tang, Melissa J. Buco, Rong N. Chang, Sandhya Dwarkadas*, Laura Z. Luan, Edward So, Christopher Ward
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

*Computer Science Department
University of Rochester

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On the Tradeoff among Capacity, Routing Hops, and Being Peer-to-Peer in the Design of Structured Overlay Networks

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 heterogeneous DHTs with $O(1)$ to $O(n)$ routing tables and identifies the good design points for typical workloads; and **(2)** proposes practical protocols to realize the potential of those good design points. We validate our protocols through both analysis and extensive simulation.

Our comparison is novel in two aspects. First, we compare totally *heterogeneous* DHTs and use *capacity* as the uniform metric to evaluate them. Second, we emphasize that a good design should strike a balance between maintenance cost and lookup cost. Our analysis shows that, for typical workloads, large routing tables actually lead to both low total 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 one-hop or two-hop routing without giving up being *peer-to-peer*. We propose 1h-Calot for one-hop routing and 2h-Calot for two-hop routing. They are purely peer-to-peer, efficient in traffic, and resilient in the face of frequent node arrivals and departures. Moreover, they are extremely simple for practical uses. 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.

1. INTRODUCTION

In recent years, Distributed Hash Tables (DHTs) have been proposed as the infrastructure for building a wide range of wide-area distributed applications such as storage [3], content distribution [2], and search engines [26]. Early DHTs [19, 22, 25, 28] 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 actually feasible to do so.

Previous works [6, 13, 14, 27] mainly compared DHTs with $O(\log(n))$ routing tables. Currently, what is lacking is an analytic framework to suggest the optimal routing table size for a given workload. This paper **(1)** compares heterogeneous DHTs with $O(1)$ to $O(n)$ routing tables and identifies the good design points for typical workloads; and **(2)** proposes practical protocols to realize the potential of those good design points. We validate our protocols through both analysis and extensive simulation.

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

1.1 Comparison of Heterogeneous DHTs

Our comparison of heterogeneous DHTs focuses on the aspects below.

- **Resilience**. DHTs target dynamic environments in which nodes may join and leave at any moment. Good designs should be able to continuously operate without disruption under these conditions.
- **Lookup latency**. Object lookup is the basic operation of DHTs. The lookup latency in DHTs needs to be low in order to accommodate interactive applications such as name resolution [18].
- **Capacity**. High capacity is desirable. It allows architects to use a smaller DHT, which is faster and cheaper, to handle a given load.
- **Being Peer-to-Peer.** A purely peer-to-peer architecture is preferable, in which nodes assume equal roles and share load evenly.

Previous works [6, 13, 14, 27] that compare DHTs with $O(\log n)$ routing tables mainly focus on the resilience and lookup latency aspects. We are among the first to emphasize the *capacity* aspect. Capacity is relevant as we expect DHTs to operate under medium to high load. Otherwise, a smaller DHT can handle the given load at a lower cost and provides better services. KaZaA [10], for instance, uses only a subset of super nodes to provide services. We measure the capacity of a DHT using the total maintenance and lookup traffic it introduces under a given workload. Reducing traffic not only saves precious bandwidth, which is the key factor that limits wide-area applications [1], but also reduces the messages processed by nodes and hence increases system capacity.

We stress the role of *workload* in the comparison. 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 a node processes per second. We are among the first to emphasize the *balance* between maintenance cost and lookup cost. In particular, *lookup rate* is a critical but historically overlooked parameter that shifts this balance. For instance, if lookups are frequent, it is beneficial to use DHTs that require high maintenance but save significantly on lookups.

Our comparison 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, for example, the argument [7] that it is favorable to maintain $O(n)$ routing tables for systems with up to a few million 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 [19, 22, 25, 28].

1.2 Non-hierarchical Protocols for One-hop and Two-hop Routing

Assuming half to several hour node lifetime (e.g., 2.9 hours in Gnutella [23]) and a medium lookup rate, 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. We emphasize that our targets are typical workloads rather than the extreme conditions assumed by some other works, e.g., only several minute node lifetime [20]. On the other hand, we also propose solutions to achieve low lookup hops under extreme dynamism, of course, at the expense of increased traffic (Section 3.2.5).

One-hop and two-hop routings are efficient in traffic and lookup hops, but their large routing tables are hard to maintain, which poses significant challenges on being resilient and peer-to-peer. Existing proposals for one-hop or two-hop routing are either hierarchical [5, 7, 8, 16, 21]—in which nodes have different roles and the load is unevenly distributed—or assume a particular query distribution that limits its generality [17]. 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 improves system resilience and eliminates the need for manual management, which are the reasons that originally gave birth to DHTs [25].

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% under typical workloads, while resolving lookups in one or two hops as opposed to $O(\log n)$ hops. Their fast lookups are particularly attractive for latency sensitive applications such as cooperative web caching and name resolution [18].

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 Calot are purely conceptual and require no explicit maintenance. 2h-Calot's randomized algorithm further exploits virtual nodes running on the same computer to connect 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, and compare them with existing DHTs. Section 5 evaluates our protocols through extensive simulation. Related work is discussed in Section 6 and Section 7 concludes the paper.

2. COMPARING HETEROGENEOUS DHTS

Previous works [6, 13, 14, 27] mainly used the resilience and lookup latency metrics to compare DHTs with $O(\log(n))$ routing tables. We instead compare heterogeneous DHTs with $O(1)$ to $O(n)$ routing tables to identify design points that have high resilience, low lookup latency, and high capacity. We are among the first to emphasize the *capacity* aspect of DHTs. High capacity is desirable. It allows architects to use a smaller DHT, which is faster and cheaper, to handle a given load.

We measure the capacity of a DHT using the total traffic it introduces under a given workload. Lower traffic leads to higher capacity. At a high level, the traffic metric captures both bandwidth consumption and DHT protocol overhead in an application independent fashion. In general, DHTs with larger routing tables introduce higher maintenance traffic but have smaller routing hops and hence lower lookup traffic. A good design should strike a balance between them to minimize the *total traffic*.

The comparison is conducted in the context of 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. Our *typical workload* assumes a node lifetime similar to that in real peer-to-peer systems. For instance, the average node lifetime in Gnutella is 2.9 hours [23]. We assume a medium to high lookup rate, $f = 0.1 \sim 10$ lookups/second. The rationale is that, if the system is underutilized, the architect should use a smaller DHT, which is cheaper and faster, to handle the given load. The self-organizing feature of DHTs makes load-driven adaptation not only possible but also beneficial [11]. Among many other real peer-to-peer systems, KaZaA [10] uses only a subset of super nodes to provide services. (This "typical workload" is a stress test for our protocols to be proposed in Sections 3 and 4. When the node lifetime is longer and the lookup rate is lower, our protocols function even better.)

We assume that node lifetime follows an exponential distribution

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

where $\lambda_l = \frac{1}{l}$. 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 average lifetime l, the node arrival rate $\lambda_e = n\lambda_l = \frac{n}{l}$. We assume the lookups that a node issues 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 retransmission 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 heterogeneous DHTs.

2.1 Degree-diameter Optimal DHTs

For a network of size n in which each node has d neighbors, the diameter D of the network is bounded [14] by: $D \geq \lceil \log_d(n(d-1)+1) \rceil - 1$. Several degree-diameter optimal DHTs [9, 12, 14, 15] approaches this lower bound. Our analysis uses de Bruijn graphs [14] as the representative, in which a lookup on average takes $r \approx \log_d n$ hops.

We calculate the *minimum 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 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. So there are $\frac{n}{l}d$ messages for node arrivals and $\frac{n}{l}d$ messages for node departures. Further, at least one message is needed to inform a new node of each of its d neighbors. So there are $\frac{n}{l}d$ messages [2] to set up the routing tables for new nodes.

Assuming all these 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. So there are nf lookups in total. Each lookup takes $\log_d n$ hops in a de Bruijn graph (or any other graphs that are close to optimal [12]). The traffic for lookups is

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

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

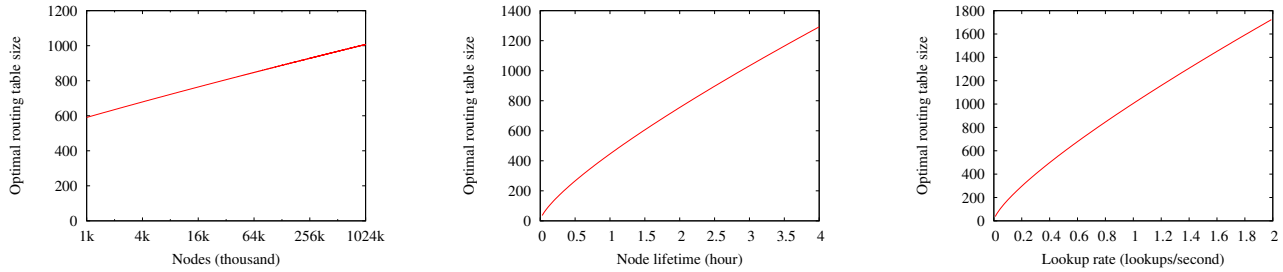
$$M = B_1 + B_2 = 1.5s \cdot (3\frac{n}{l}d + nf \log_d n). \tag{5}$$

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

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

[1] In most existing DHTs, nodes probe their routing neighbors periodically. An "ideal" design can avoid this traffic. For instance, Calot uses overlay multicast, instead of probing, to maintain routing tables (see Section 3).

[2] The traffic would be lower if the new node copies a complete routing table from an existing node in a single, big packet. This only affects our analysis results by a small constant factor. We choose not to consider this optimization here because it is adopted in few existing DHTs. The protocol specific analysis in Section 3 and 4 will consider details like this.



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

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

The fl component in Equation 6 indicates that the optimal routing table size d is proportional to the number of lookups a node processes in its entire lifetime. Previous comparisons mainly focus on node lifetime's impact on system resilience and totally ignore lookup rate. Our analysis instead shows that lookup rate is a critical parameter when designing systems for high capacity.

Equation 6 has no closed form solution. We solve it using the 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 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. One-hop routing and two-hop routing are the good design points we focus on.

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 [14] based on de Bruijn graph. Other systems that do use large routing tables are based on a hierarchy [5, 7, 8, 16, 21]. In Sections 3 and 4, we will present our peer-to-peer solutions.

2.2 One-hop Schemes

One-hop schemes maintain a complete routing table on each node, $d = n - 1 \approx n$. Substituting this into Equation 5, we obtain the total traffic

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

2.3 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

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

2.4 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 [25], 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 2.1, 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 is

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

2.5 Unstructured Overlay Networks

For unstructured overlay networks such as Gnutella, we optimistically assume that they do not incur any maintenance traffic and queries are flooded to every node in the system. The traffic to process nf queries is

$$M_{flood} = (1 + 0.5)s \cdot n^2 f. \tag{10}$$

2.6 Comparing Traditional DHTs with Others

In Figure 2, we compare the traffic of traditional DHTs with that of one-hop schemes, two-hop schemes, and unstructured overlay networks. Overall, Figure 2(a) and (b) show that one-hop and two-hop schemes can be traffic efficient and low latency at the same time. 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 it is not economical in traffic to do so for systems with more than 2,000 nodes. With a few million nodes, one-hop schemes could introduce 1,000 times more traffic than traditional DHTs. Figure 2(b) shows that "ideal" two-hop schemes can be traffic efficient for systems with up to tens of millions of nodes. Figure 2(c) suggests that query flooding schemes used in unstructured overlay networks are more efficient in traffic than traditional DHTs only if nodes have very short lifetime (about 75 seconds) and lookups are very infrequent (one lookup every 30 minutes).

2.7 Proactive vs. Reactive Maintenance

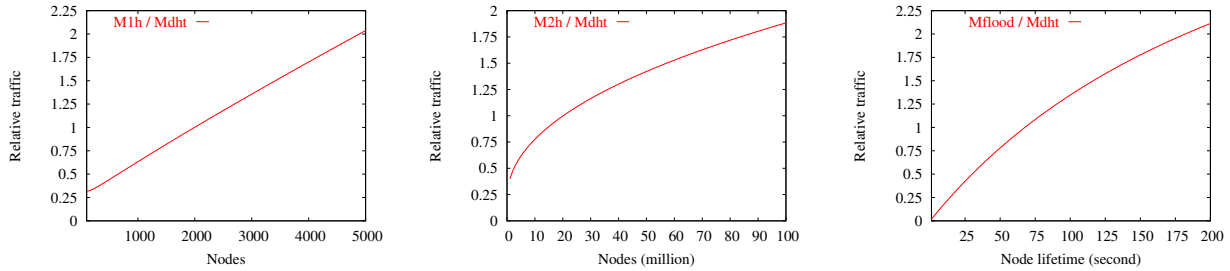
All DHTs described above proactively maintain the accuracy of the routing tables. An alternative is reactive maintenance, in which nodes cache other nodes they discovered in past lookups and reuse them in future lookups. There is no explicit maintenance operation. The drawback is that nodes may encounter frequent failures during lookups. Next, we calculate the probability of correct cache hits 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 node N issues follow a Poisson process with rate f. Since there are n nodes, the lookups that node N issues to target node S is a Poisson process with rate $\lambda_v = f/n$. The interval between lookups from N to S follows an exponential distribution

$$p_v(t) = \lambda_v e^{-\lambda_v t}. \tag{11}$$

Node lifetime follows the exponential distribution $p_l(t)$ in Equation 1. When node N contacts node S at time x since the last lookup, the probability that S is still alive is $(1 - \int_{y=0}^{y=x} p_l(y)\ dy)$. So the probability that node N finds node S is still alive when N issues a new lookup to S is

$$\begin{aligned} P_{cache\_hit} &= \int_{x=0}^{x=\infty} [\ p_v(x)\ (1 - \int_{y=0}^{y=x} p_l(y)\ dy)\ ]\ dx \\ &= \frac{\lambda_v}{\lambda_v + \lambda_l} = \frac{1}{1 + \frac{n}{lf}}. \end{aligned} \tag{12}$$

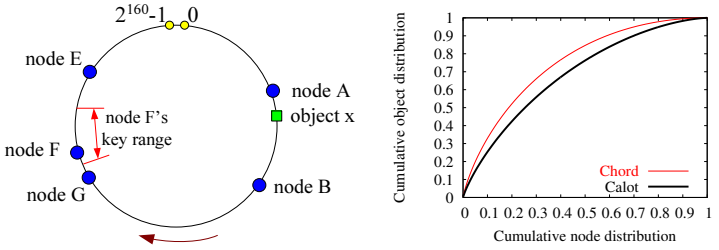


(a) One-hop schemes. (l=2.9 hours, f=0.1 lookups/second)

(b) Two-hop schemes. (l=2.9 hours, f=0.1 lookups/second)

(c) Unstructured overlay networks. (n=1,000 nodes, $f=\frac{1}{1800}$ lookups/second)

Figure 2: *Traffic relative to traditional DHTs that use routing tables of size $O(\log n)$. M_{1h}, M_{2h}, M_{dht}, and M_{flood} are from Eq. 7-10.*



(a) Mapping objects to nodes.

(b) Object distribution.

Figure 3: *Difference between Calot and Chord. **(a)** Calot stores an object on the node whose identifier is the closest to the object's key in* absolute *distance. Object x is stored on node A, because A is the closest to x. Chord would store object x on node B, because B immediately follows x clockwise. **(b)** The cumulative distributions of nodes and objects when storing one million objects into a 1,000-node Calot and a 1,000-node Chord, respectively. The load in Calot is more balanced.*

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 hops. A failed hop introduces long 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 and omit the analysis for its traffic.

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 replicate complete $O(n)$ routing tables on every node. For systems with up to a few thousand nodes, 1h-Calot resolves lookups in one hop with high probability while introducing traffic lower than traditional DHTs. 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, 16, 21], 1h-Calot and 2h-Calot are purely peer-to-peer. Below we simply refer to 1h-Calot as *Calot* when the context is clear.

3.1 Overview

Like Chord [25], Calot organizes nodes into a circular ring that corresponds to identifier space [0, 2^{160}-1] (see Figure 3). 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* (see below) 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). By contrast, Chord stores an object on the node whose identifier follows the object's key clockwise. Figure 3(a) depicts the difference. Calot balances object distribution across nodes better than Chord (see Figure 3(b)). In Calot, a node owns a small key range only if it is close to both its predecessor and its successor. In Chord, a node owns a small key range if it is close to its predecessor.

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 case that the routing tables are inaccurate (e.g., missing living nodes or including 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 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). The maintenance of predecessors and successors is done through periodic heartbeat messages. Except for its predecessor and successor, a node 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.

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

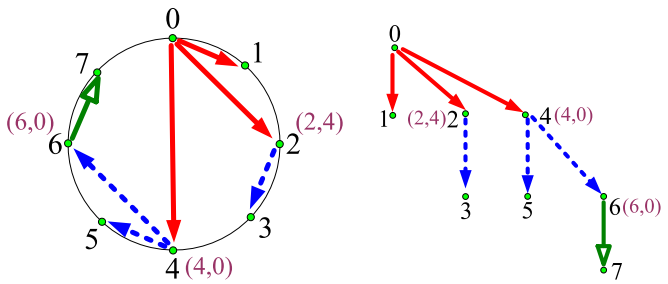
Table 1: *Cache hit rate in Equation 12 (n=1,000 and l=2.9 hours).*

3.2 Routing Table Maintenance

3.2.1 Handling Node Arrivals

We assume that a new node N knows through out-of-band channels about at least one node P already in the system. Node N generates a random identifier k by applying SHA-1 hashing to its IP address, and then asks node P to route a message to locate the predecessor and the successor of k. Node N joins the ring topology assuming k as its identifier and takes over from its predecessor and successor objects that are closer to N. Node N copies a complete routing table from its predecessor in order to have a global view of the system.

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 4). Unlike approaches [5, 7] that use a single pre-determined hierarchy to propagate events, Calot multicasts each notification through a *different* tree expanded *just in time* according to the current content of the routing tables. In different trees, a node sits at different levels. Overall,



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

Figure 4: *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 joins 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 children of node 0 is responsible for covering a range of the identifier space, for instance, 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. Calot uses a total of n trees. The "trees" are conceptual and require no explicit maintenance.*

the load is evenly distributed across all nodes. Our approach retains the true spirit of the peer-to-peer architecture.

We first give some definition before describing the process for nodes to select their children in 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 [25], with high probability, each node has $O(\log_2 n)$ distinct finger nodes (note that, for instance, the successor nodes of keys k_0 and k_1 may be the same node since the two keys are close).

The new node N sits at the root of the multicast tree to announce its arrival. Among nodes in its routing table (which is obtained from its predecessor), node N selects its finger nodes as its children. Let S_i and s_i $(i = 1, \cdots, m)$ denote the m 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 [3] consisting of N's identifier, N's IP address, and a multicast range (s_i, s_{i+1}) of the identifier space. Node S_i will be responsible for multicasting the notification to nodes whose identifiers are in range (s_i, s_{i+1}). Together, the m finger nodes of node N help N to multicast its arrival to all nodes in the system.

Node S_i uses a process similar to what node N uses to expand the multicast tree with its own children. But the purpose now is to cover nodes in range (s_i, s_{i+1}) rather than the entire identifier space. 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 multicast 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 multicast tree by adding their finger nodes as children, and so forth. A node stops expanding the tree when it discovers that there is no node in the multicast range it is assigned to, i.e., the multicast range is sufficiently small.

Unlike those pre-determined hierarchies [5, 7], the multicast "trees" in Calot are transient and purely conceptual. There is no message to construct the trees before use; no probing to maintain the trees (see Section 3.2.4); 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 and inconsistent routing tables. (See Section 3.2.2 for the details on handling failures.) 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 children 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 down. In Calot, nodes maintain accurate predecessors and successors through heartbeats.

[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 hashing of the IP addresses.

3.2.2 *Handling Node Departures*

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 arrival. A node N may fail without a notice. Its predecessor detects this through lost heartbeats and then announces node N's departure.

The average session duration in Gnutella is 2.9 hours [23], 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 failure or software failure. 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 to keep the routing tables up to date.

3.2.3 *Handling Failures*

Without faults, each node receives a membership change notification through a multicast tree exactly once. Faults, however, are unavoidable in distributed environments. There are several scenarios that 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 $Q_1, \cdots, Q_m$. If node P is no longer in the system, which may happen when the routing table of node S is inaccurate, S will timeout due to missing acknowledgment from node P. Node S deletes node 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 node P dies after receiving and acknowledging the notification but before forwarding it to nodes $Q_1, \cdots Q_m$, these nodes will miss this notification.

Inaccurate routing tables do not persistently fail lookups so long as nodes properly maintain their predecessors and successors. But inaccurate routing tables degrade routing performance by taking more hops (when living 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 unacceptable routing performance. To this end, we propose node re-announcement to address the problem of missing living nodes and routing entry timeout to address the problem of stale dead nodes.

When a 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 opportunities to pick 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

$$\int_0^h p(t)dt = \frac{1}{2} \implies h = l \ln 2 \approx 0.7l. \tag{13}$$

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 the routing table after the timer fires. Therefore, the number of dead nodes in a routing table does not accumulate over time.

In summary, with timeout and re-announcement, routing tables become *soft-state* images of the system. When the system stabilizes and no faults

happen over a certain period, the routing tables converge to a correct global view. The overhead, however, is the traffic for re-announcements. We next calculate this overhead. A living node re-announces its existence every h seconds (see Equation 13) and half nodes leave before they make the first re-announcements. So half 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.

3.2.4 Multicast Traffic Concentration

Some approaches [5, 7] give up being peer-to-peer and use pre-determined hierarchies to propagate events. Among the $n(n-1)$ links between n nodes, these systems use only a fixed set of $n-1$ links in the hierarchy. Calot disseminates notifications through n different trees built from the $O(n \log n)$ links between nodes and their finger nodes. Another extreme solution is to use almost all $n(n-1)$ links in the n trees. We believe our choice strikes a good balance between load sharing and efficiency.

Regardless of which of the n trees is in use for the current notification, a node in Calot always forwards the notification to a subset of its $O(\log n)$ finger nodes. Thanks to this traffic concentration, a node talks to its finger nodes more frequently, which helps detect dead finger nodes faster and reduce timeout and retrials in multicast. In other words, the concentrated traffic itself serves as a form of piggybacked "probes". (Recall that, in Calot, nodes only send heartbeats to their predecessors and successors.) The frequency of the "probes" automatically adjusts to the dynamism of the system. As node lifetime becomes shorter, there are more multicast notifications and nodes automatically "probe" their finger nodes more frequently. We next calculate the effect of the piggybacked "probes"

A node on average initiates three multicast notifications during its lifetime: one for its birth, one for its death, and one re-announcing its existence. Amortized over node lifetime l, a node receives q=$\frac{3n}{l}$ notifications per second (e.g., when n=1,000 and l=2.9 hours, q=0.29). Each notification travels through $n-1$ links. In total, Calot uses $n \log_2 n$ distinct links in the n multicast trees. On average, the number of notifications that travel over a link per second is

$$\lambda_g = \frac{\frac{3n}{l}(n-1)}{n \log_2 n} \approx \frac{3n}{l \log_2 n}. \quad (14)$$

We assume that the notification traveling over a link is a Poisson process with rate λ_g. Suppose a link connects a node P and its finger node Q. Node Q may leave at any moment between two notifications traveling over the link. Node Q's lifetime follow an exponential distribution with rate $\lambda_l = \frac{1}{l}$ (see Equation 1). Following exactly the same analysis process that leads to Equation 12, we know that, when node P sends its finger node Q a notification, the probability that P finds Q dead is

$$P_{dead} = \frac{\lambda_l}{\lambda_l + \lambda_g} \approx \frac{1}{1 + \frac{3n}{\log_2 n}}. \quad (15)$$

The probability P_{dead} is independent of node lifetime l, which corroborates our intuition that the frequency of the piggybacked "probes" automatically adjusts to the dynamism of the system. Thanks to the "probes", the probability of encountering dead nodes during a multicast is low, for instance, P_{dead}=0.003 when n=1,000. The longest path in the multicast tree has $O(\log(n))$ hops and the average path length is $\frac{\log_2(n)}{2}$. The reliable and fast propagation of notifications guarantees the accuracy of the routing tables. P_{dead} decreases as the system size n increases, indicating that the effect of piggybacked "probes" is scalable.

Equation 15 is just an approximation. The probability that a node N communicates with a dead finger node P during a multicast is actually lower than P_{dead}. This is because node N may have already received the departure announcement of node P before N tries to communicate with P. In this case, node N simply chooses another node as the finger node to replace P. Even if node N finds its finger node P dead in the multicast process, it does not give up; it chooses another node S to replace P and asks S to forward the notification. Also note that Equation 15 only estimates the *average* of λ_g. Although the traffic is evenly distributed across nodes, the traffic over different links vary dramatically. Links that connect nodes and their short distance finger nodes carry higher traffic.

A good estimation of the accuracy of the routing tables needs to consider the factors listed above as well as other factors such as timeout and re-announcement, network delay, and probably changing system size (e.g., a system whose size grows gradually). In Section 5, we use simulation to evaluate a complete system that consider all these factors. Our results show that the routing tables are maintained accurately and most lookups are resolved in one hop.

In addition to the benefits of piggybacked "probes", concentrating multicast traffic on the $O(n \log n)$ links between nodes and their finger nodes also provides the opportunity for notification aggregation. When the frequency of notifications is beyond a node's forwarding capability, it can aggregate notifications received over a time period, say 5 seconds, and then send them in a single packet to its finger nodes. We recommend notification aggregation only as the last resort, because it increases notification delay and leads to less accurate routing tables.

3.2.5 Handling Extreme Dynamism

For typical workloads in which nodes have half to several hour node lifetime (e.g., 2.9 hours in Gnutella [23]), Calot is able to maintain accurate routing tables using low traffic. As the dynamism of the system increases, the accuracy of the routing tables degrades. This is because more nodes fail during the multicast process. For these extremely dynamic conditions, there are two directions to improve 1h-Calot: parallel multicast or parallel lookup. Both come at the cost of increased traffic.

Parallel multicast propagates a notification through multiple trees concurrently. For instance, when a new node N joins, in addition to announcing its arrival through the tree rooted at itself, N also randomly chooses j other nodes S_i (i=1, $\cdots$, j) and asks S_i to disseminate a notification through the trees rooted at S_i. We call this method "redundant trees". Another alternative is "redundant flooding", which floods a notification through all the $n \log n$ links between nodes and their finger nodes (recall that Calot uses these links to construct the n multicast trees). With this method, a node that receives or initiates a notification simply sends the notification to each and every of its $O(\log n)$ finger nodes. Nodes remember the notifications they forwarded lately to avoid forwarding the same notification repeatedly. Each node receives a notification for $O(\log n)$ times from different incoming links. Our simulation shows that "redundant flooding" is more reliable than "redundant trees" that use $O(\log n)$ trees, although they introduce the same level of traffic. This is because the former covers more links than the latter.

Parallel multicast improves the accuracy of the routing tables. Parallel lookup, on the other hand, tries to work with inaccurate routing tables. The initiator N of a lookup searches its routing table for j nodes that are the closest to the lookup key, and send them requests in parallel. Each of the j nodes processes the lookup normally, either returning the object or forwarding the lookup greedily.

Parallel multicast and parallel lookup can be used either independently or together to handle extreme dynamism. For typical workloads, we recommend the basic Calot without these extensions because it is more efficient in traffic and already maintains very accurate routing tables.

3.3 Traffic Analysis

In this section, we compare the total traffic in Calot with that in traditional DHTs [19, 22, 25, 28]. Results in Section 5 show that Calot maintains accurate routing tables and resolves most lookups in one hop. 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

$$L_o \approx (1 + 0.5)s \cdot nf. \quad (16)$$

Next, we calculate the traffic for maintenance. Each second, $\frac{n}{l}$ new nodes join. We estimate 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

$$M_{o1} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (17)$$

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

$$M_{o2} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (18)$$

Each new node obtains a complete n-entry routing table from its predecessor. 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 routing tables 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

$$M_{o3} = 0.25s \cdot \frac{n}{l} n. \quad (19)$$

Each second, $\frac{n}{l}$ nodes leave the system. The notifications for node departures contain 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

$$M_{o4} = (1 + 0.5)s \cdot \frac{n}{l} n. \quad (20)$$

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

$$M_{o5} = 0.5s \cdot \frac{2n}{T}. \quad (21)$$

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

$$\begin{aligned} M_o &= L_o + M_{o1} + M_{o2} + M_{o3} + M_{o4} + M_{o5} \\ &= s \cdot n(1.5f + \frac{1}{T} + \frac{4.75n}{l}). \end{aligned} \quad (22)$$

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

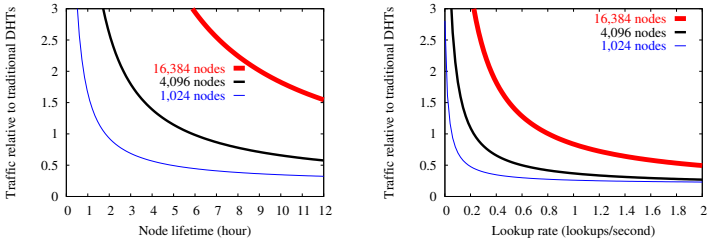
$$R_o = \frac{M_o}{M_{dht}} \approx \frac{6.3}{fl} \cdot \frac{n}{\log_2 n}. \quad (23)$$

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 by the savings from efficient one-hop lookups. As a result, 1h-Calot actually introduces less total traffic than traditional DHTs. The relative traffic R_o between 1h-Calot and traditional DHTs 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 schemes [5, 7]. Membership update traffic in one-hop schemes grow quadratically with the system size, due to more frequent membership changes in large systems and the fact that each change is notified to more nodes.

Figure 5 plots the relative traffic R_o in Equation 23. 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, one hop as opposed to $O(\log n)$ hops.

4. 2H-CALOT FOR TWO-HOP ROUTING

When the system size 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 Equation 7 and 22). 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) on page 4 shows that, even for very large systems, the total traffic in "ideal" two-hop schemes is still lower than traditional DHTs under typical workloads. Moreover, two-hops schemes resolve lookups in two hops, much faster than traditional DHTs.



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

Figure 5: *Relative traffic between 1h-Calot and traditional DHTs (the exact R_o in Equation 23).*

Our goal, therefore, is to design a practical two-hop protocol that approaches the performance of "ideal" two-hop schemes. The main challenge is to maintain the large $O(\sqrt{n})$ routing tables in the face of frequent node arrivals and departures 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 Protocol Description

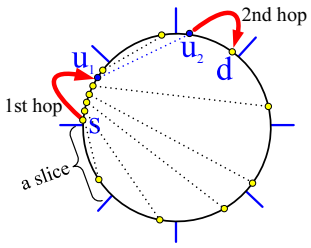
4.1.1 The Basic Version of 2h-Calot

2h-Calot is a further development of 1h-Calot. It also organizes nodes into a ring topology that corresponds to identifier space [0, 2^{160}-1]. The "basic" version of 2h-Calot partitions the ring into continuous regions of equal size called *slices* (see Figure 6(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 deliveries 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 runs two virtual nodes called *sister nodes*. 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.

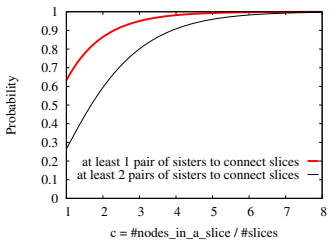
Suppose there are a large number of nodes S_i (i=1,$\cdots$,j) in a slice $\mathcal{S}$. The sister nodes P_i of nodes S_i are randomly distributed all over the identifier space because the node identifiers are randomly generated. Given an arbitrary destination slice $\mathcal{D}$, with high probability, one of these sister nodes P_i may sit in slice $\mathcal{D}$. In other words, with high probability, we can find a pair of sister nodes to connect slices $\mathcal{S}$ and $\mathcal{D}$. Please refer to Figure 6 for a detailed illustration.

We next derive a proper configuration for 2h-Calot. Let k denote the total number slices, m denote the number nodes in each slice, and n denote the number of computers. $k \cdot m = 2n$ since each computer runs two virtual nodes. 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

$$\text{node/slice ratio:} \quad c = \frac{m}{k} \quad (24)$$



(a) Illustration of 2h-Calot.



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

Figure 6: *Highlights of 2h-Calot.* *2h-Calot organizes nodes into a ring topology. Each computers runs two virtual nodes (nodes at the two ends of a dashed link, e.g., nodes u_1 and u_2), and we refer to them (e.g., u_1 and u_2) as* sister nodes. *Each node joins the overlay independently as if it were a "real" node. The identifiers of nodes are randomly generated. The basic version of 2h-Calot partitions the identifier space into slices. Inside a slice, everyone knows everyone else through a membership multicast protocol similar to 1h-Calot. If there are a large number of nodes in each slice, given two random slices $\mathcal{S}$ and $\mathcal{D}$, we can ensure with high probability that there exist a pair of sister nodes u_1 and u_2 that connect $\mathcal{S}$ and $\mathcal{D}$. This is because the sister nodes of nodes in a slice are randomly distributed all over the identifier space. Suppose node s in slice $\mathcal{S}$ wishes 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 $\mathcal{D}$. In summary, sister nodes act as gateways to route messages between slices in one hop. 2h-Calot is purely peer-to-peer. Section 4.1.2 further explains how to make slices adaptive.*

be the main configuration parameter for 2h-Calot. c is a *constant* that defines the ratio between the number of nodes in a slice and the total number of slices. Since $c = \frac{m}{k}$ and $k \cdot m = 2n$, we have

$$\text{nodes in a slice:} \quad m = \sqrt{2cn} \tag{25}$$

$$\text{total slices:} \quad k = \sqrt{\tfrac{2n}{c}}. \tag{26}$$

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

$$p = \frac{1}{k}. \tag{27}$$

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:

$$P(X = x) = \binom{m}{x} p^x (1-p)^{m-x} \approx e^{-c} \cdot \frac{c^x}{x!} \tag{28}$$

$$P(X \geq 1) = 1 - P(X{=}0) \approx 1 - e^{-c} \tag{29}$$

$$P(X \geq 2) = 1 - P(X{=}0) - P(X{=}1) \approx 1 - e^{-c} - ce^{-c}. \tag{30}$$

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

$P(X \geq 1)$ is the probability that there exist 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 6(b) plots $P(X \geq 1)$ and $P(X \geq 2)$. This figure shows that, with high probability, we can find sister nodes to connect two random slices. Therefore, the two-hop routing in Figure 6(a) can be accomplished. We choose to use the configuration when $c = m/k = 5$.

$$c = 5 \implies m = \sqrt{10n}, \quad k = \sqrt{0.4n}, \tag{31}$$

$$P(X \geq 1) \approx 0.993, \quad P(X \geq 2) \approx 0.960 \tag{32}$$

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 the 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.

4.1.2 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 k of slices, making it impossible to do decentralized adaptation based on only local knowledge. Below we show how to make 2h-Calot adaptive as the system grows or shrinks.

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*). Further, 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*). So 2h-Calot can do 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 6(a). Unlike the fixed, global slices, each node has its own neighbor zone centered at itself and nodes 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. So 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 doest not distinguish if P is from its "neighbor set" or its "sister set". Nodes have no notion of "slices" either. The only rule is greedy forwarding. With high probability, the first hop of a lookup forwards the lookup to a node P that is in the "sister set" of the query initiator and is very close to the destination; the second hop directly delivers the lookup to the destination node, which is a node in node P's "neighbor set".

4.1.3 Routing Table Maintenance

In this section, we describe the algorithm for routing table maintenance. It is similar to the one for 1h-Calot but with a major difference: when a node N joins or leaves, the multicast notification is only sent to N's $\frac{m}{2}$ 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 node arrival as 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 by the $y+2$ neighbors (including nodes P and N_j) is z. Node N_j estimates the total number of computers in the system as

$$\text{estimated total computers:} \quad n = \frac{1}{2}\frac{2^{160}}{z}(y+2). \tag{33}$$

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

$$\text{estimated neighbor zone:} \quad \mathcal{K} = [\, d - \tfrac{1}{2}b, d + \tfrac{1}{2}b\,]. \tag{34}$$

(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 in the estimation is well tolerated. With high probability, a node can forward messages in one hop to any region in the identifier space through the sisters of nodes in its neighbor zones.

Nodes 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 4): node N_j selects its forward-finger nodes within the forward range $\mathcal{K}_f$ as its children; each child P of node N_j will help N_j to cover a sub-range $\mathcal{K}_p$ of range $\mathcal{K}_f$ by asking its forward-finger nodes within the sub-range $\mathcal{K}_p$ to forward the notification, and so forth. The multicast process in tree T_b that covers the backward range K_b is the same as that in tree T_f except that the multicast travels over links between nodes and their backward-finger nodes.

2h-Calot shares many features with 1h-Calot. For instance, parallel multicast and parallel lookup can be used to handle extreme dynamism. Like 1h-Calot, entries in the routing tables time out periodically and living nodes re-announce their existence periodically. Before a re-announcement, a node always re-estimates the system size n based on current information in its routing table and updates its estimation of neighbor zone $\mathcal{K}$ accordingly. The re-announcement will cover nodes in the updated neighbor zone. This helps nodes with long lifetime to adapt as the system evolves.

4.1.4 2h-Calot vs. Other Two-hop Schemes

Existing solutions 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* solutions that make 2h-Calot particularly attractive for practical use.

- 2h-Calot is extremely simple and purely peer-to-peer. Nodes know their m neighbors along the ring. *That's it!* Notification multicast uses these m neighbors; routing also uses these m neighbors. There are neither real "slices" nor real multicast "trees" to maintain; both are conceptual. By contrast, existing *hierarchical* solution [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.
- 2h-Calot distributes load evenly across nodes. Each pair of sister nodes carry some traffic between two "slices". By contrast, for each slice, existing solutions select a few nodes to act as gateway to carry all incoming traffic from other slices. The load is unevenly distributed across nodes.
- 2h-Calot estimates neighbor zones from local knowledge and adapts as the system evolves. By contrast, existing solutions use fixed slices. All nodes need to agree upon the slice boundaries that are determined at design time, and therefore cannot adapt easily.

4.2 Protocol Analysis

In this section, we compare the total traffic in 2h-Calot with that in traditional DHTs [19, 22, 25, 28]. The analysis 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.

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

$$L_t \approx (1 + 0.5)s \cdot 2nf. \tag{35}$$

Next, we calculate the traffic for maintenance. Each second, $\frac{2n}{l}$ new nodes (or $\frac{n}{l}$ computers) arrive. Conceptually, the notification for a node arrival 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 have 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 nodes exactly once. The traffic to multicast notifications for node arrivals is

$$M_{t1} = (1 + 0.5)s \cdot \frac{2n}{l}m. \tag{36}$$

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

$$M_{t2} = (1 + 0.5)s \cdot \frac{2n}{l}m. \tag{37}$$

Each new node copies a routing table from its predecessor. Conceptually, the routing table includes m neighbors and the sisters of those m neighbors. But 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 routing tables 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

$$M_{t3} = 0.25s \cdot \frac{2n}{l}m \tag{38}$$

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

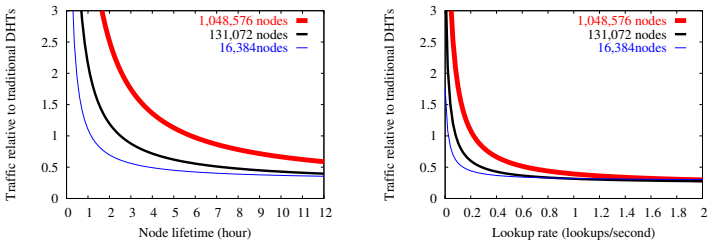
$$M_{t4} = (1 + 0.5)s \cdot \frac{2n}{l}m. \tag{39}$$

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

$$M_{t5} = 0.5s \cdot \frac{4n}{T}. \tag{40}$$

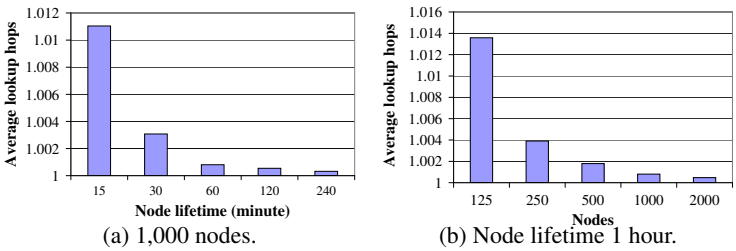
The total traffic (maintenance plus lookup) for 2h-Calot is

$$\begin{aligned} M_t &= L_t + M_{t1} + M_{t2} + M_{t3} + M_{t4} + M_{t5} \\ &= s \cdot n(3f + \frac{2}{T} + \frac{9.5\sqrt{10n}}{l}). \end{aligned} \tag{41}$$



(a) (b)

Figure 7: *Relative bandwidth consumption between 2h-Calot and DHTs. (a) Vary node lifetime (lookup rate f=0.1). (b) Vary lookup rate (node lifetime l=2.9 hours).*



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

Figure 8: *Routing hops per lookup in 1h-Calot (the Y axis stars from 1).*

Comparing Equations 22 and 41, we see that 2h-Calot is more scalable than 1h-Calot. 2h-Calot's traffic grows with $n^{1.5}$ while 1h-Calot's traffic grows with n^2.

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

$$R_t = \frac{M_t}{M_{dht}} \approx \frac{12.7\sqrt{10n}}{fl\log_2 n}. \quad (42)$$

Like 1h-Calot, the traffic for 2h-Calot is dominated by membership updates. The maintenance traffic, however, is well compensated by the savings from efficient two-hop lookups. As a result, under typical workloads, 2h-Calot still introduces less total traffic than traditional DHTs.

Figure 7 plots the exact R_t in Equation 42. 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, two hop as opposed to $O(\log n)$ hops.

When the load (lookup rate) is below medium, 2h-Calot introduces more traffic than traditional DHTs. The is because the maintenance traffic is not sufficiently compensated by the savings from the lookup traffic. When the system is underutilized, it is not necessary to use that many nodes to provide the services. We can dynamically downsize the system to provide more responsive and more predictable services. KaZaA [10], for instance, uses only a subset of super nodes to provide services.

1h-Calot and 2h-Calot are just the level-1 and level-2 of our Calot family. Generally, a level-i Calot uses level-(i-1) Calots as building blocks and resolves lookups in i hops. In a level-i Calot, each computer runs 2^{i-1} virtual nodes to connect remote "super slices". Regardless of the level of a Calot, it is still purely peer-to-peer as opposed to hierarchical. Higher level Calots are more efficient in traffic as the maintenance traffic grows with $O(n^{\frac{1}{i}})$. On the other hand, we consider 1h-Calot and 2h-Calot sufficient for most applications.

5. EXPERIMENTAL RESULTS

We built an event-driven simulator to evaluate 1h-Calot and 2h-Calot. The simulator consists of 8,100 lines of C++ code and runs on Linux. It simulates a complete system, including dynamic node arrival and departure, timeout, and network delay. We do not simulate the UDP/IP level packet details in order to scale to thousands of nodes. 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 (32,000 virtual nodes).

Modeling network topology and latency is still an open research area. We follow the approach [6] that focuses on ensuring the simulated network latency 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 [4], which is extracted from real measures 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 add 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 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 use them to evaluate the routing performance, during which each node on average issues 1,000 random lookups. We choose to report the lookup performance at the end of the simulation rather than the lookup performance averaged over the entire simulation time. This choice reports lookup performance more pessimistically because the routing tables are less accurate at the end of the simulation compared with the time the system just started.

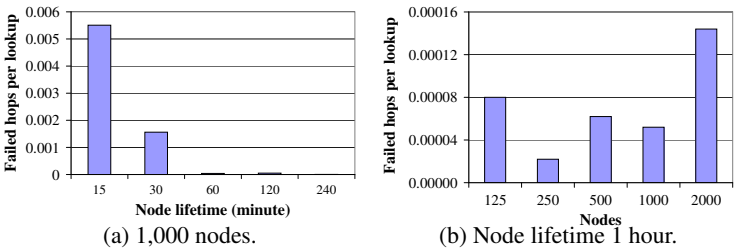
Our simulator models the lookups that a node issues during its lifetime as a Poisson process with rate f. However, unless related statistics are needed, the simulator does not fully execute the lookups issued before the evaluation phase. We found this optimization important to make the simulation time manageable when the system size is large. This optimization makes the reported lookup performance more pessimistic because the overlooked lookups can actually help detect and fix some inaccurate entries in the routing tables.

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

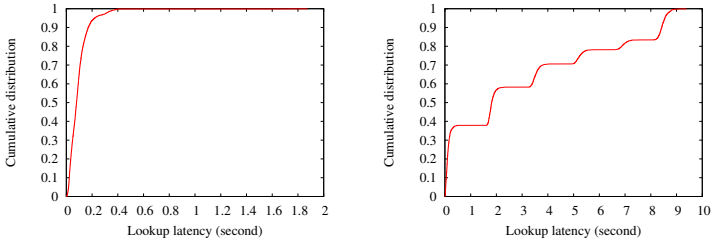
We first present results on 1h-Calot. Figure 8 shows the average routing hops per lookup when varying node lifetime and system size. In Figure 8(a), the routing performance improves as node lifetime increases. With one hour lifetime, the average routing hops is only 1.0008, indicating that the routing tables are very accurate. In Figure 8(b), the routing performance improves as the system size increases, which seems counter-intuitive but is actually consistent with our analysis in Section 3.2.4 (see Equation 15). As the system becomes larger, there are more multicast notifications and nodes communicate with their finger nodes more frequently. Disruptions in multicast due to dead finger nodes happen at a lower rate. The notifications reach nodes faster and more reliably, which leads to more accurate routing tables and improved routing performance.

Figure 9 reports the average number of failed hops encountered per lookup. (Note that a failed hop does not necessarily lead to a unresolvable lookup. The system always retries alternative routing paths.) Both missing living nodes and stale 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 9(a) shows that the failed hops w reduce dramatically as node lifetime increases. With half hour lifetime, w=0.0016; with one hour lifetime, w=0.000052 (about 1 failed hop out of 20,000 lookups). In Figure 9(b), the number of failed hops fluctuates due to the random-



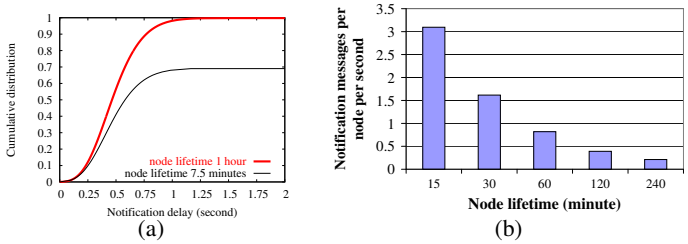
(a) 1,000 nodes. (b) Node lifetime 1 hour.

Figure 9: *Failed routing hops per lookup in 1h-Calot.*



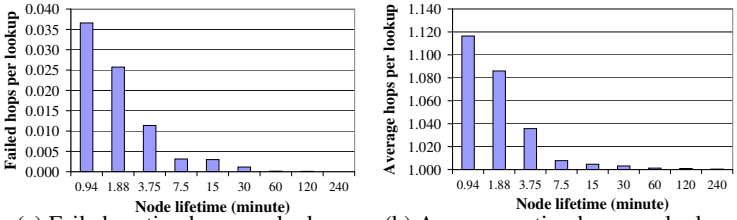
(a) Node lifetime 1 hour. (b) Node lifetime 7.5 minutes.

Figure 11: *CDF of the lookup latency for a 1,000-node 1h-Calot.*



(a) (b)

Figure 10: *A 1,000-node 1h-Calot. (a) Delivery delay of multicast notifications. (b) The number of notifications that a node receives per second.*



(a) Failed routing hops per lookup (b) Average routing hops per lookup

Figure 12: *A 1,000-node 1h-Calot, using "redundant flooding" to handle extreme dynamism. A notification is sent through $O(n \log n)$ links.*

ness inherent in the simulation and the fact that the number of failed hops is not monotonic with respect to system size (an approximate analysis is omitted due to space limitation).

Figure 10(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 happens. This quick and reliable distribution of membership changes is the key that 1h-Calot can maintain accurate routing tables. When the node lifetime time 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 nodes receive the notification within 2 seconds (we did not show nodes that receive the notification after 2 seconds). Figure 10(b) reports the average number of notification messages a node receives per second. With 2 hour node lifetime (2.9 hours in Gnutella [23]), 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 aggregate notifications for the same outgoing link and send them in a single packet.

Figures 8 and 9 show that the average routing performance is good. In Figure 11, we plot the distribution of the lookup latency in a 1,000-node system. With one hour node lifetime, the 95 percentile lookup latency is only 220ms. For extremely short node lifetime (7.5 minutes), the lookup latency is much higher. The saw-like curve is due to the timeout and retrials that happen for multiple times during a lookup.

In Section 3.2.5, we proposed several methods to handle extreme dynamism. Figure 12 reports the results for the "redundant flooding" method, which floods each notification through all $O(n \log n)$ links between nodes and their finger nodes. Each node receives a notification for up to $O(\log n)$ times. This redundancy improves reliability and also delivers notifications faster because the delivery delay is the minimum delay of the redundant paths. This figure shows that, with "redundant flooding", 1h-Calot can achieve good routing performance even when the node lifetime is very short. When the lifetime is longer than or equal to half hour, the improvement in routing performance over the basic 1h-Calot is not significant because the basic 1h-Calot already maintains very accurate routing tables. We recommend "redundant flooding" for applications that need low-latency lookups in very dynamic environments. Again, nodes can aggregate notifications when the message rate is high.

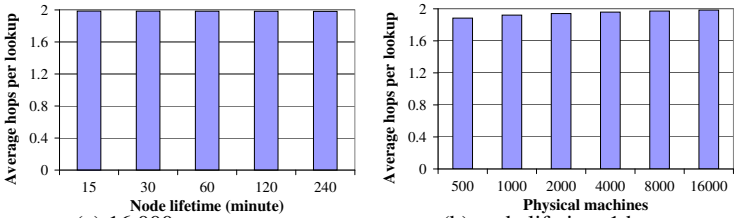
5.2 2h-Calot

1h-Calot and 2h-Calot share many features. Due to space limitation, our evaluations of 2h-Calot will focus on the aspects unique to 2h-Calot.

Figure 13 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. Comparing with 1h-Calot, 2h-Calot's performance in this figure 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 13(a). Further, 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 13(a), the number of nodes in one tree is 200, equivalent to a small 1h-Calot system. Due to the limitation of 2GB memory, 16,000 computers are the largest size we can simulate.

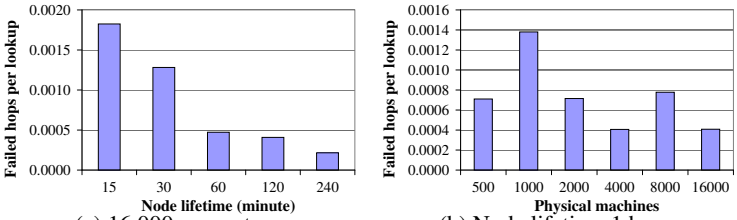
Figure 14 plots the failed routing hops per lookup. Like 1h-Calot, 2h-Calot maintains very accurate routing tables. As a result, the failed hops are very low. With 16,000 computers and one hour node lifetime, it encounters only one failed hop out of 2,000 lookups. Like 1h-Calot, the failed hops in Figure 14(b) fluctuate as the population grows, due to the randomness inherent in the simulation and the fact that failed hops is not monotonic with respect to system size. Comparing Figure 14 and 9, 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 Figure 15 and 16, 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 add 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 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 fluctuation still exists due to the randomness. The arrival rate λ_e stay at that level until the population reaches $(1 + \frac{1}{10})n$ for the first



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

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



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

Figure 14: *Failed routing hops per lookup in 2h-Calot.*

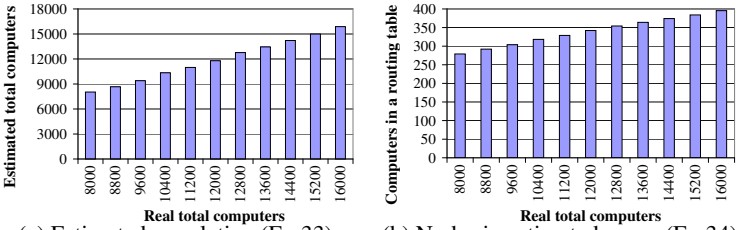
time. Then the arrival rate is increased again to $\lambda_e = (1 + \frac{2}{10})\frac{n}{l}$ and stay at that level until the population reaches $(1 + \frac{2}{10})n$. Generally, the arrival rate stays at level $\lambda_e = (1 + \frac{i}{10})\frac{n}{l}$ $(i = 1, \cdots, 10)$, until the population reaches $(1 + \frac{i}{10})n$. The simulation ends when the population reaches $2n$. In 21 simulated hours, the population 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 33 to estimate the total number of computers and use Equation 34 to estimate their neighbor zones. Figure 15 plots estimated total computers and the number of computers that nodes keep in their routing tables (i.e., computers in nodes' estimated neighbor zones). Both are averaged over all nodes and are presented as functions of the growing system size. Despite the fact that 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 consistent views 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-hops schemes [7, 16, 21] cannot. Figure 16 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.

Our last experiment evaluates 2h-Calot's sensitivity to its only major parameter c, which decides the number of nodes in each 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 28, the probability of resolving lookups in two hops is 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 high probability and also gives some buffer zone to handle the dynamism as the population grows

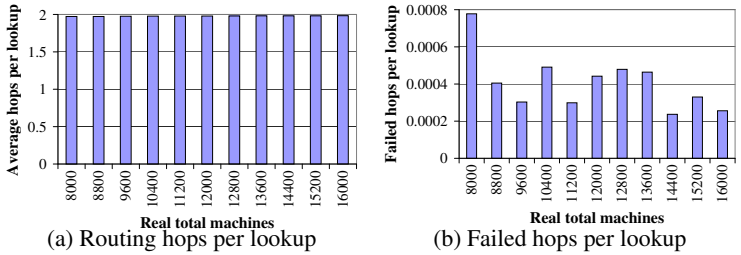
$c = m/k$	1	2	3	4	5	6	7
avg. hops	2.431	2.131	2.035	1.999	1.984	1.976	1.972

Table 2: *Routing hops per lookup in a n=16,000 machine system while varying the parameter c that decides the number of nodes in each neighbor zone ($m = \sqrt{2cn}$).*



(a) Estimated population (Eq 33). (b) Nodes in estimated zones (Eq 34)

Figure 15: *Stress test for 2h-Calot's adaptation ability as the population grows from 16,000 to 32,000 computers. The node lifetime is one hour.*



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

Figure 16: *Stress test for 2h-Calot's adaptation ability as the population grows from 8,000 to 16,000 computers. Average node lifetime is one hour.*

or shrinks.

6. RELATED WORK

This paper makes contributions in suggesting the optimal routing table size that maximizes the capacity of DHTs and in the designs of purely peer-to-peer one-hop and two-hop protocols.

Recent works have extensively compared DHTs with routing tables of size $O(\log n)$ [6, 13, 14, 27]. We instead introduce capacity as the uniform metric to compare DHTs with $O(1)$ to $O(n)$ routing tables. Xu et al. [27] studied the tradeoff between routing table size and network diameter. Several degree-diameter optimal DHTs have been proposed [9, 12, 14, 15]. Our work is built upon these works. We intend to answer the question of "given many degree-diameter optimal DHTs, what's the routing table size that maximizes the system capacity 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. The ring-topology overlay is partitioned into slices and each slice is further partitioned into units. Each slice has a leader and each unit also has a leader. When a membership change happens, a node sends a notification to its slice leader, which forwards the notification to all other slice leaders. A slice leader forwards a received notification to unit leaders in its slice, which propagate the notification to ordinary nodes. Unlike our peer-to-peer 1h-Calot protocol, nodes in this scheme have different roles and run different protocols. Slice leaders have much higher load than others and the system critically relies on them to function. Gupta et al. recommended this protocol 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. Gupta et al. also proposed a hierarchical two-hop scheme. Similarly, the ring is partitioned into slices and units. Inside a slice, everyone knows everyone else. The slice leader of a slice selects some representative nodes in that slice to carry all incoming traffic to that slice. See Section 4.1.4 for a detailed comparison between this protocol and 2h-Calot.

Beehive [17] replicates objects according to object popularity to achieve $O(1)$ lookups. There are applications in which the short lifetime of objects makes them unsuitable for replication; and there are applications that have no objects to replicate at all, for instance, message indirection [24].

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 gossip to disseminate membership changes. Gossip is not efficient in traffic because a node may receive the same notification for 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. [16] 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 gossip to propagate membership information. Rodrigues et al. [21] proposed a system for one-hop routing. It is based on the client-server architecture. Well-provisioned special servers inform other nodes of the system configuration.

Compared with the works above, we believe 1h-Calot and 2h-Calot are the first purely *peer-to-peer* one-hop and two-hop protocols.

7. CONCLUSIONS

In this paper, we compared DHTs with $O(1)$ to $O(n)$ routing tables and suggested that, for typical workloads, large routing tables actually lead to both low total 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. We proposed 1h-Calot and 2h-Calot for one-hop and two-hop routing, respectively. They are purely peer-to-peer, efficient in traffic, and resilient in the face of frequent node arrivals and departures. Both 1h-Calot and 2h-Calot are extremely simple: multicast maintains the routing tables; information in the routing table 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.

We made the following contributions in this paper.

- We are among the first to compare heterogeneous DHTs with routing tables varying from $O(1)$ to $O(n)$. We are also among the first to emphasize the capacity aspect of DHTs and use it as the the uniform metric to evaluate heterogeneous DHTs.

- We are among the first to emphasize the balance between maintenance cost and lookup cost. We show that *lookup rate* is a critical but historically overlooked parameter that shifts this balance.

- We are among the first to suggest that large routing tables actually lead to both low total traffic and low lookup latency.

- We proposed 1h-Calot. It is purely peer-to-peer. By contrast, existing one-hop protocols are hierarchical. 1h-Calot multicasts node arrivals and departures through $O(n)$ different trees embedded in the overlay. Unlike existing overlay multicast, the "trees" in 1h-Calot are purely conceptual and require no explicit maintenance.

- We proposed 2h-Calot. It is adaptive and purely peer-to-peer. By contrast, existing two-hop protocols are hierarchical and use fixed slices that cannot adapt. 2h-Calot's randomized algorithm exploits virtual nodes running on the same computer to connect remote "slices".

The low traffic and low hop features of 1h-Calot and 2h-Calot make them attractive for most DHT applications, especially those that operate under constant high load and require fast routing. We plan to use 1h-Calot and 2h-Calot in our large-scale monitoring framework, which provides real-time usage and performance data about software license, CPU, network, and storage to facilitate job scheduling and anomaly detection in advanced on-demand computing environments [29].

8. REFERENCES

[1] C. Blake and R. Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *HotOS'03*, May 2003.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *SOSP,* 2003.

[3] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP'01*, October 2001.

[4] 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.

[5] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. HiScamp: self-organising hierarchical membership protocol. In *European 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'03*, August 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'03*, Berkeley, CA, USA, February 2003.

[9] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. Berkeley, CA, USA, February 2003.

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

[11] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. M. Vahdat. Using Random Subsets to Build Scalable Network Services . In *USITS*, 2003.

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

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

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

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

[16] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage. Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In *Proceedings of the IEEE Workshop on Internet Applications*, 2003.

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

[18] V. Ramasubramanian and E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *ACM SIGCOMM*, 2004.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM'01*, 2001.

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

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

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[23] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)*, 2002.

[24] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirec infrastructure. In *ACM SIGCOMM'02*, 2002.

[25] 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'01*, 2001.

[26] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM'03*, 2003.

[27] J. Xu, A. Kumar, and X. Yu. On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks. *IEEE Journal on Selected Areas in Communications*, 22(1):151–163, Jan 2004.

[28] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, 2001.

[29] IBM On Demand Business^{TM}. http://www.ibm.com/ondemand.