EUROPEAN TRANSACTIONS ON TELECOMMUNICATIONS
Euro. Trans. Telecomms. 2004; **15**:571–587
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/ett.1013

Special Issue on P2P Networking and P2P Services

Ulysses: a robust, low-diameter, low-latency peer-to-peer network†

Abhishek Kumar[1]*, Shashidhar Merugu[1], Jun (Jim) Xu[1], Ellen W. Zegura[1] and Xingxing Yu[2]

[1]*College of Computing, Georgia Institute of Technology, Atlanta, GA 30032, USA*
[2]*School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30032, USA*

SUMMARY

A number of distributed hash table (DHT)-based protocols have been proposed to address the issue of scalability in peer-to-peer networks. In this paper, we present Ulysses, a peer-to-peer network based on the butterfly topology that achieves the theoretical lower bound of $\log n/\log\log n$ on network diameter when the average routing table size at nodes is no more than $\log n$. Compared to existing DHT-based schemes with similar routing table size, Ulysses reduces the network diameter by a factor of $\log\log n$, which is 2–4 for typical configurations. This translates into the same amount of reduction on query latency and average traffic per link/node. In addition, Ulysses maintains the same level of robustness in terms of routing in the face of faults and recovering from graceful/ungraceful joins and departures, as provided by existing DHT-based schemes. The performance of the protocol has been evaluated using both analysis and simulation.

1. INTRODUCTION

Recent years have seen a considerable amount of research effort devoted to the development of distributed and coordinated protocols for scalable peer-to-peer file sharing (e.g. [1–5]). These protocols generally rely on distributed hash tables (DHTs) to allow each node to maintain a relatively small routing table, while taking a relatively small number of overlay routing hops to route a query to the node responsible for the particular DHT key. DHT-based protocols have great potential to serve as the foundation for a wide-range of new internet services (e.g. [6, 7]).

A fundamental question in DHT design is the tradeoff between the routing table size, i.e. number of neighbors a DHT node has to maintain, and the network diameter, i.e. the number of hops a query needs to travel in the worst case. It was observed in Reference [8] that existing DHT schemes tend to have either (1) a routing table of size $O(\log_2 n)$ and network diameter of $O(\log_2 n)$ (including References [1–4]), or (2) a routing table of size d and network diameter of $O(dn^{1/d})$ (including Reference [5]). It was asked in Reference [8] whether $\Omega(\log_2 n)$ and $\Omega(dn^{1/d})$ are the asymptotic lower bounds for the network diameter, when the routing table sizes are $O(\log_2 n)$ and d respectively. Recent work [9] shows that neither is lower bound: the actual bounds are $\Omega(\log_2 n/\log_2\log_2 n)$ and $\Omega(\log_d n)$ respectively.

An open question posed in Reference [9] concerns the design of a DHT-based scheme that achieves the lower bounds without causing stress, where stress is defined informally as requiring certain nodes and edges to forward traffic that is many times the average, even under uniform load. The main contribution of this paper is Ulysses‡ a stress-free DHT-based protocol based on a butterfly topology that meets the theoretical lower bounds. In particular, with a routing table size of about $\log_2 n$, Ulysses can reach the diameter of $\lceil\log_2 n/\log_2\log_2 n\rceil + 1$, as compared to $\log_2 n$ in most existing schemes with the same routing table

* Correspondence to: Abhishek Kumar, 250 14th Street, GCATT, Room 220, Atlanta, GA 30318, USA. E-mail: akumar@cc.gatech.edu
† A preliminary version of the paper was presented at IEEE ICNP 2003, held in Atlanta, GA, U.S.A., November 4th to 7th, 2003. The paper has been significantly expanded and revised for journal submission.
‡ We briefly introduced Ulysses, in about one page, in our prior work [9]. However, Ulysees is used in Reference [9] only as a counterexample to a theoretical conjecture. It is by no means a careful and thorough exploration of Ulysses. We feel that the overlap of this work with [9] is minimal.

Received 5 July 2004
Revised 5 July 2004
Accepted 14 July 2004

size.[§] This represents a significant reduction in network diameter, especially for large network size n. For example, in a network of 1 million ($\approx 2^{20}$) nodes, the average diameter of Chord[¶] [1] is 20, but that of Ulysses is 6, with similar average routing table sizes (20 for Chord and around 17 for Ulysses). For larger values of n, the advantages of Ulysses become even more pronounced (e.g. 30 hops vs. seven hops when $n = 10^9$). Such a saving has a magnified impact at the network layer because each hop in a peer-to-peer network is an application-layer overlay hop.

Ulysses not only achieves a reduction in network diameter (the worst-case query latency), by a factor of $\log_2 \log_2 n$, but also achieves a reduction factor of $(1/2) \log_2 \log_2 n$ on the average query latency, compared to Chord [1]. By Little's Law [10], this reduction results in significant reduction in the overall network traffic in the network as each query 'travels less'. Therefore, with the same number of nodes and links and the same offered load (queries), each Ulysses link or node will have to carry less traffic than in Chord.

Ulysses is based on the well-known butterfly network structure [11]. The desired properties do not come immediately, however, and require adapting the butterfly structure for use in practical peer-to-peer networks by addressing three primary challenges. First, the basic butterfly has edge stress where some edges carry significantly more traffic than average. To address this, we add edges to the butterfly to avoid edge stress while maintaining small routing table size. Second, we must embed the butterfly structure onto a dynamically changing set of peers. Third, since peer-to-peer networks are subject to considerable dynamics as nodes join and leave, we must provide for procedures that allow routing to stabilize upon joins and leaves, and provide for correct routing while stabilization is occurring.

The remainder of the paper is structured as follows. We give an overview of our design objectives and the Ulysses approach in Section 2. After introducing the static butterfly topology and its shortcomings in Section 3, we describe Ulysses in detail in Section 4. We present a verification of our design using formalism from distributed computing in Section 5. In Section 6, we discuss performance evaluation results of Ulysses. Section 7 surveys other related work followed by concluding remarks in Section 8.

[§]In Section 7, we discuss some recently proposed schemes that achieve the bound of $O(\log_2 n / \log_2 \log_2 n)$.

[¶]For comparison, we have chosen Chord as a representative of the whole family of DHT based protocols that have a network diameter of $O(\log n)$ and routing table size of $O(\log n)$.

2. DESIGN OVERVIEW OF ULYSSES

In this section, we first describe three major design objectives of Ulysses. Then we give a brief overview of how these objectives are achieved in Ulysses.

2.1. Design objectives

2.1.1. Low latency routing. Ulysses tries to achieve the lowest possible network diameter with a routing table size of approximately $(\log n)$ per node, without causing excessive stress at any overlay link or node. While the butterfly topology can help us achieve this in theory, it remains a challenging task to embed the butterfly structure in a dynamic network so that two important properties are satisfied. First, each query, starting from any location, should be able to find its way to the destination within the same small number of steps as in a butterfly network. Second, the routing table size should be approximately $(\log n)$ at each node. Both properties need to be maintained despite node joins and departures.

2.1.2. Self-stabilization. The system should be able to recover to a 'correct state' after joins and leaves temporarily 'perturb' the system. The recovery should not only be fast and inexpensive, but also handle multiple concurrent faults that are possible in a dynamic peer-to-peer environment.

2.1.3. Robustness. The protocol must include the ability to route a query around faulty nodes to reach a non-faulty destination. This capability is orthogonal to self-stabilization: self-stabilization says that the network will eventually recover from faults whereas the robustness says that most of the network should function despite the existence of transient faults. Also, from the performance point of view, such 'detours' should not increase the query latency too much, even when a large percentage of nodes are faulty in the network.

2.2. How Ulysses achieves these objectives

The multidimensional static butterfly topology, described in the next section, is known to have a diameter of $O(\log n / \log \log n)$ with an out-degree of $\log n$ at each node. However, past attempts to use this topology in DHTs have only used the constant out-degree (say d) version of this topology that achieves a diameter of $O(\log_d n)$ [12]. Ulysses is the first to propose and use an enhanced version of this topology to achieve a diameter of $O(\log n / \log \log n)$ with average out-degree of $(\log n)$. After proposing the enhanced version of the butterfly topology in the next

section, we show in Section 4 how to embed this topology in a dynamic peer-to-peer network (with joins/departures), to achieve the objective of low latency routing.

Ulysses has a set of self-stabilization mechanisms to handle joins as well as graceful and ungraceful departures of nodes. Our embedding of the enhanced butterfly topology, combined with a 'zoning' scheme that partitions the key-space among different nodes is designed to accommodate such dynamics. We describe these in Subsections 4.5 and 4.6. A salient feature of Ulysses' self-stabilization mechanisms is their low message complexity, discussed further in Subsection 4.7.

Ulysses is designed to be robust. The routing mechanism contains an implicit 'detour protocol' that redirects the query to a different node when a fault is encountered on the routing path. We also solve a 'vicious cycle' problem associated with the naive detour protocol. With this refined detour protocol, the system can continue to operate despite a large number of faulty nodes. Our simulation result shows that even under severe cases where 20% of nodes are rendered inoperable, the system can still serve 99.95% of the queries destined for non-faulty nodes.

3. THE STATIC BUTTERFLY TOPOLOGY

The structure of Ulysses is inspired from the well-known butterfly topology [11]. In this section, we first describe the static butterfly, which has low diameter but is not a practical peer-to-peer network because it requires a precise number of nodes at each level, functioning correctly at all times for correct routing. Another problem with the static butterfly is that some of the edges carry a disproportionately high amount of traffic as compared to the average, or in other words, have a high edge stress. We propose a solution to this problem, which is subsequently used in the Ulysses network as explained in Section 4.

3.1. The static butterfly topology and its 'stress' problem

The general static butterfly network can be defined as follows. A (k, r)-butterfly is a directed graph with $n = kr^k$ vertices, where k and r are referred to as the diameter and the degree respectively. Each vertex is of the form $(x_0, x_1, \ldots, x_{k-1}; i)$, where $0 \leqslant i \leqslant k-1$ and $0 \leqslant x_0, x_1, \ldots, x_{k-1} \leqslant r-1$, i.e. $x_0, x_1, \ldots, x_{k-1}$ are base r digits. For each vertex $(x_0, x_1, \ldots, x_{k-1}; i)$, we refer to i as its level[‖] and $(x_0, x_1, \ldots, x_{k-1})$ as its row. From each vertex $(x_0, x_1, \ldots, x_{k-1}; i)$, there is a directed edge to all vertices of the form $(x_0, x_1, \ldots, x_i, z, x_{i+2}, \ldots, x_{k-1}; i+1)$ when $i \neq k-1$ and $(z, x_1, \ldots, x_{k-1}; 0)$ when $i = k-1$, where $z = 0, 1, 2, \ldots, r-1$. The routing from vertex $(x_0, x_1, \ldots, x_{k-1}; i)$ to vertex $(y_0, y_1, \ldots, y_{k-1}; j)$ proceeds in two phases. In the first phase, x_{i+1} successively changes to y_{i+1} while going from level i to level $i+1$, x_{i+2} to y_{i+2} while going from level $i+1$ to level $i+2$, and so on. This process continues until all of the x's have been changed to y's, and then in the second phase, we continue along row $(y_0, y_1, \ldots, y_{k-1})$ to level j.

Figures 1(a) and (b) show a $(2, 2)$ static butterfly drawn in two different ways. Notice that in Figure 1(b) the outgoing edges fan out, first along dimension 1 while going from level 0 to level 1, and then along dimension 0 while going from level 1 to level 0. This fanout corresponds to the wildcard z discussed earlier.

Definition 1—Adapted From Reference [9]. *We say that a network is edge stress free if under the assumption of uniform traffic load,[**] the amount of traffic going through any edge in the network is no more than c times the average. Here $c \geqslant 1$ is a small constant.*

The static (k, r) butterfly is not *edge-stress free*. Consider the edges going from a node $(x_0, x_1, \ldots, x_{k-1}; i)$ to $(x_0, x_1, \ldots, x_{k-1}; i+1)$, i.e. the edges between nodes with the same row identifier but different levels. We call such edges as horizontal links. Non-horizontal links are called cross links. In the static (k, r) butterfly, each node has exactly one horizontal link and $(r-1)$ cross links. A query traverses k cross links on average in the first phase of routing and $(k-1)/2$ horizontal links in the second phase of routing as described above. Therefore, a horizontal link carries about $(r/2)$ times as much traffic as a cross link.

3.2. Shortcut links to remove stress

We solve this stress problem by adding $k-2$ 'shortcut' links from the node $(x_0, x_1, \ldots, x_{k-1}; i)$ at level i to nodes $(x_0, x_1, \ldots, x_{k-1}; j)$ at level $j, \forall j \in \{0, 1, \ldots, k-1\}, j \neq i, i+1$. This way, in the aforementioned butterfly routing, once x's have all changed to y's in the first phase of routing, only one jump is needed in the second phase to reach the destination through one of these 'short-cut' links. This not only eliminates the edge stress, but also has the additional

[‖]Throughout this paper, it is assumed that additive operations on level are modulo k.

[**]Since the hashing scheme in a DHT based network distributes the keys uniformly across the key space, the queries are uniformly distributed across nodes in the network. If in addition, the sources of queries are themselves assumed to be distributed uniformly, the resulting scenario is one of uniform traffic load.

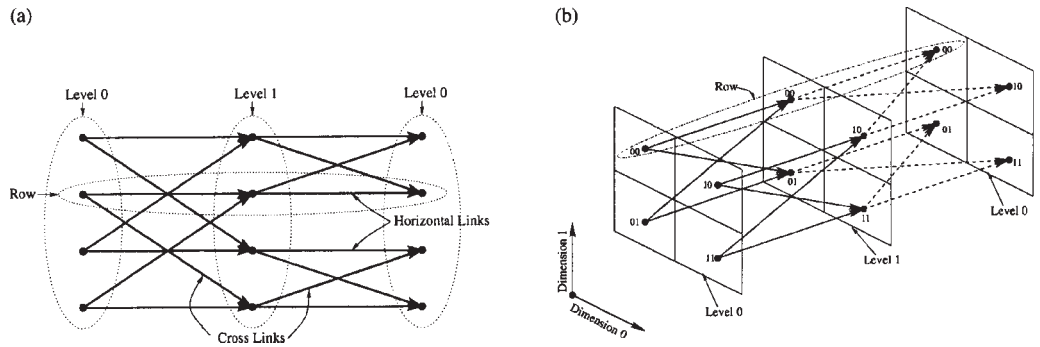

Figure 1. (a) A $(2,2)$ static butterfly topology with two levels and each node with degree $= 2$ illustrating rows, horizontal and cross links. It is 'wrapped around' with level 0 drawn twice. (b) An alternative representation of the same $(2,2)$ static butterfly topology, with nodes and their identifiers. Solid edges from level 0 to level 1 are along dimension 1 and dashed edges from level 1 to level 0 are along dimension 0. An eight-node peer-to-peer network can be represented by this $(2,2)$ butterfly topology. The square around each node represents the zone of responsibility of the node. The total area of all the squares in both the levels represents the entire DHT space.

benefit of reducing the network diameter from $2k-1$ to $k+1$. Recall that the number of nodes n is related to the number of levels k through the equation $n = kr^k$. If we choose k to be $(\log n/\log\log n)$, we get $r = (n/(\log n/\log\log n))^{(1/(\log n/\log\log n))} < \log n$. Thus the increase in the routing table size due to the shortcut links is from $(r = \log_2 n)$ to $(r + k - 2 = \log_2 n + \log_2 n/\log_2 \log_2 n - 2)$. For example, when there are 2^{20} nodes in the network, this represents an increase from 20 to 23 entries in the routing table.

This topology is not yet suitable for a peer to peer network because it requires a precise number of nodes at each level, for correct routing. The Ulysses network uses a modified version of this topology, which can accommodate the dynamics of a peer-to-peer network, as shown in the following section.

4. DESIGN OF THE ULYSSES PROTOCOL

In this section, we present the detailed design of Ulysses that achieves the objectives outlined in Section 2. We first present its naming and zoning scheme, which concerns the allocation of different portions of the DHT key-space to nodes. We then describe the topology of the Ulysses network and the details of routing, including robustness. We end this section with a description of the self-stabilization mechanism of Ulysses, which handles node joins and departures, and enhancements that achieve better topology control without incurring extra overheads.

4.1. Naming and zoning in Ulysses

In Ulysses, we use a naming convention that is different from the one used above in the static butterfly. This naming convention retains the essential notions of row and level while providing a mechanism that captures the details of allocation of portions of the hash table to different nodes as well as the dynamics of node joins and departures.

4.1.1. Naming convention. In the Ulysses network with k levels and n nodes, each DHT node represents a zone in the name-space, and is identified by a tuple (P, l). Here P is a binary string, also known as the row identifier and l is the level, where $0 \leqslant l \leqslant k-1$. The correspondence between the row identifier P of a Ulysses node and the k-dimensional row identifier $(x_0, x_1, \ldots, x_{k-1})$ in a static butterfly is as follows: the bits at location $i, i+k, i+2k \ldots$ in P correspond to x_i. In other words, the coordinate of the node (P, l) in the ith dimension, is given by the concatenation of every kth bit in the string P starting at the ith location. For example if $k = 5$ and $P = a_0a_1 \ldots a_{12}$, then $a_2a_7a_{12}$ is the coordinate of P in the second dimension (i.e. x_2 in the row identifier of a static butterfly). The expected length of row identifier P of a node in a Ulysses network with n nodes and k levels is $\log_2(n/k)$. However, the actual length of P for individual nodes varies due to dynamic arrival and departure of nodes, as explained later in this section. The concept of level in Ulysses is the same as in static butterfly.

In Ulysses, the search key is also mapped to a tuple (α, l) using one or more uniform hash functions. Again, l corresponds to the level. The row identifier α is m bits long, where m is a constant chosen such that $k \times 2^m$ is large enough to assign unique keys for all objects stored in the DHT. In particular, the number of keys is much larger than the number of nodes, i.e. $k \times 2^m \gg n$, which is an assumption implicit in all DHT based schemes (e.g. [1–5]).

4.1.2. Distributing the hash table. The structure of Ulysses is closely related to the way Ulysses nodes are allocated portions of the DHT. A node with identifier (P, l) stores all keys (α, l) such that P is a prefix of α. In this manner, the key-space is partitioned into disjoint zones of responsibility, with one DHT node handling each zone. A key (α, l) belongs to a zone (P, l) if and only if P is a prefix of α. For example, the keys (1011001, 3) and (1011100, 3) would belong to the zone (1011, 3). From the description of node joins and departures, to come later in this section, it will become clear that no two nodes P and P' exist in the same level l such that P is a prefix of P'. This means that a query will be handled by one and only one node.

Here is an intuitive, geometric way of understanding Ulysses' design. The key-space of a Ulysses k-butterfly can be seen as k different k-dimensional level-cuboids,[††] one level-cuboid corresponding to each level. Each level-cuboid is of size $2^{m/k}$ in each dimension. A search key (α, l) maps to a specific point in the lth level-cuboid. The nodes in each level partition the level-cuboids into smaller zone-cuboids, as shown in Figure 2.

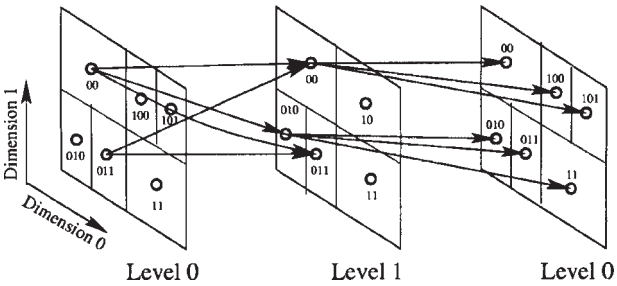

Figure 2. A Ulysses butterfly with two levels and 11 nodes. Unlike Figure 1(b) where all zones of responsibility are homogenous, the arrival and departure dynamics of nodes in the peer-to-peer network cause non-uniformity in the size of the zones of responsibility (areas of regions around the nodes). Links from only two nodes in each level are shown for clarity. It is 'wrapped around' with level 0 shown twice. If we project the zone of node {011,0} on level 1 and slide its 'shadow' along the vertical direction (dimension 1), it overlaps with zones {011,1} and {00,1}. This explains why node {011,0} has links to nodes {011,1} and {00,1}.

4.2. Topology of the Ulysses network

Having specified the zones of responsibility for nodes in the Ulysses network, we proceed to define the links between these nodes. The topology of the Ulysses network captures the link structure of the static butterfly described earlier in Subsection 3.1. In particular, we wish to retain the property that links from nodes in level i to those in level $i+1$ fan-out along the dimension $i+1$. The shortcut links that were introduced in Subsection 3.2 to make the network edge stress free and reduce its diameter, are also retained in Ulysses.

In Ulysses, the links can be geometrically visualized as follows: each zone-cuboid (P, l) in the level l has links to all those zone-cuboids $(P', l+1)$ in the level $(l+1)$ such that P' has an overlap with P in all dimensions other than the dimension $i+1$. Since, we do not place the requirement of an overlap in the dimension $i+1$, the links fan-out along this dimension. For example, in Figure 2, the node 00 at level 0 has links to the three nodes 00, 010 and 011 in level 1, because their zones have an overlap with the zone of node 00 along all dimensions other than dimension 1. Thus the links from node 00 at level 0 fan out along dimension 1.

Another way to visualize this process is the following: project the 'shadow' zone of the node (P, l) on level $l+1$ and stretch it along the dimension $l+1$. (P, l) has links to all nodes $(P', l+1)$ whose zones overlap with this stretched shadow. Returning to Figure 2, if we project the zone of node {011,0} onto level 1 and stretch this 'shadow' along the vertical direction (dimension 1), it overlaps with zones {011,1} and {00,1}. This explains why node {011,0} has links to nodes {011,1} and {00,1}. These fan-out links correspond to the horizontal and cross links in the butterfly topology.

Additionally, each node (P, i) also has links to all nodes (P', j) such that either P is a prefix of P' or *vice versa*, irrespective of the value of j. These links correspond to the 'shortcut' links in the butterfly topology. The geometric intuition here is that a zone/node will have a link to all zones overlapping with its 'shadow' at all levels.

4.3. Routing in a Ulysses butterfly

Routing in the Ulysses network can be visualized as follows. A query for the key (α, i) originates at some random

[††] A cuboid is the generalization of a cube in higher dimensions.

node (P, l) in the network. In the first step, (P, l) forwards this query to the node $(P', l+1)$ such that the location of α in dimension $l+1$ matches the range of the zone P' in dimension $l+1$. After this forwarding, we say that $(P', l+1)$ 'locks' on the destination α along the dimension $l+1$. Then, the node $(P', l+1)$ forwards this query to $(P'', l+2)$ such that the location of α in dimensions $l+1$ and $l+2$ lies within the range of the zone P' in dimension $l+1$ and $l+2$. Thus in each step, the query gets locked in one additional dimension, and after the first k steps the query reaches a node (Q, l) such that α lies within the zone Q in all the k dimensions. If the level l is the same as the level i of the key that is being searched, then Q must contain α, and the routing is done. Otherwise the node (Q, l) forwards the query on the shortcut link to node (Q, i), which must be responsible for the key (α, i).

The pseudo-code for the forwarding operation at a node is shown in Figure 3. At a node (P, l) in the Ulysses network, routing of a query for the key (α, i) proceeds as follows. First the set S of dimensions in which the query has already been locked is computed locally by examining the bits of P and α (line 5). If the query has been locked in all dimensions and the level of this node is the same as the level of the key that is being queried for, then the routing is done (line 7). If all the dimensions have been locked but the levels do not match, the query is forwarded on the shortcut link to the correct level (line 8). If one or more dimensions do not match, the query is forwarded to the next level locking dimension $l+1$, while maintaining the lock on the dimensions in S (line 9). Since the forwarding operation at any node depends only on the destination node address, it is stateless. We present a formal proof of the correctness and complexity analysis of routing in Subsection 5.1.

4.4. Robustness

To add robustness to the routing mechanism, we add an extra phase to the routing algorithm to handle the situation where a neighbor $(P', l+1)$ has been identified as the correct next hop for a query but this neighbor does not respond to attempts to forward the query to it. In such a situation, the current node (P, l) needs to 'route around' this failed neighbor. If this failed node was the node responsible for the key being queried, then the query is considered failed since nothing can be done before the fault has been repaired (discussed in Subsection 4.6). Otherwise the current node should try to circumvent this failed neighbor by forwarding the query to one of its other neighbors while maintaining the condition that the dimensions in S that have been already locked, remain locked after this step. Usually there are many such neighbors and the query is forwarded to one of them randomly, resulting in a 'random offset' to the query path along the next dimension instead of locking it in (line 15). The query will get locked in the remaining dimensions and reach

```
1. Routing algorithm at a node (P,i)
2. input:  a query key (α,j)
3. output:  a ''next-hop'' decision
4. begin
5.    Compute the set of dimensions S on which P has locked α
6.    if (S == Ω) then /* all dimensions have been locked */
7.       if (i == j) then destination has been reached;
8.       else forward the packet through a shortcut link to the destination, at level j;
9.    else forward the packet to a neighbor (P',i+1) that locks α on the set S⋃{i+1};
10.   /* existence of such a neighbor guaranteed by Lemma 2 */
11.   if (the ''next-hop'' is faulty in line 8 or line 9)
12.      if (S == Ω − {i+1}) then
13.         if (i+1 == j) then nothing can be done; /* the destination is faulty */
14.         else trigger the ''vicious cycle avoidance protocol''; /* described in Section
4.4.1 */
15.      else forward the packet to a random neighbor (P',i+1) that locks α on S;
16. end
```

Figure 3. Algorithm for routing at a node, without the optimization in Subsection 4.4.2.

back to the current level at another node (Q, l), after additional $k-1$ steps. Note that because the query has been locked in other dimensions, both (Q, l) and the next-hop calculated by it are different from (P, l) and $(P', l+1)$ respectively. Thus the dimension $l+1$ will be locked correctly on the 'second attempt'.

4.4.1. 'Vicious cycle' avoidance. A special case arises when the current node (P, l) is at level l, and all dimensions other than $l+1$ have been locked. The node $(P', l+1)$ in the next level, to which this query would be forwarded in normal course, has failed. Attempts to route around this failed node result in a 'vicious cycle' as follows. After (P, l) gives the query path a random offset in dimension $l+1$, the query simply propagates back on $k-1$ horizontal links to some other node (Q, l) in the level l. Although (Q, l) is different from (P, l), it independently calculates the next hop neighbor as $(P', l+1)$ again. Thus the query continues to fail to get locked even after receiving multiple random offsets along dimension $l+1$. To avoids this 'vicious cycle', a node whose next-hop neighbor for a query is faulty, first checks if the query has been locked in all other dimensions. If so, it forwards this query to a random neighbor at level $l+1$, requesting this neighbor to provide a random offset at level $l+2$. It can be shown that this effectively avoids the vicious cycle problem.

4.4.2. An optimization. Shortcut links can also be used in cases where the query is already locked in the next two or more dimensions, thus reducing the length of the routing path. In this case, a node can directly forward the query on its shortcut link to the level j such that $j+1$ is the first level (in a cyclic manner) that needs to be locked. Although this optimization does not reduce the worst case latency (in number of hops), it does reduce the average latency, especially for queries that have to be re-routed due to failures on their paths.

4.5. Self-stabilization on join of a new node

The following two operations need to be performed when a new node joins Ulysses.

4.5.1. Finding a row. A new node $\mathcal{N}$ that would like to join the network first randomly generates a key (α, l) and sends a query for this key, through a node X that is already in the Ulysses network. To find such an X, the new node can use any of the discovery mechanisms proposed in the literature, such as Reference [13]. This query, routed through the Ulysses network, will eventually reach the node $\mathcal{O}$ with identifier (P, l) currently responsible for the key (α, l). Node $\mathcal{O}$ then splits its zone of responsibility in two and assigns one-half to the new node $\mathcal{N}$ as follows. $\mathcal{O}$ changes its own row identifier to '$P||0$' and assigns the new node $\mathcal{N}$ the row identifier '$P||1$'. Both nodes remain in level l. Here '||' denotes string concatenation. The keys that are stored at the node $\mathcal{O}$ but are better matched now with the new node $\mathcal{N}$'s identifier are handed over to $\mathcal{N}$. The nodes $\mathcal{N}$ and $\mathcal{O}$ are referred to as buddies. Formally $P = \text{buddy}(P')$ if and only if P and P' differ in only their last bit. This simple mechanism, to choose a zone that should be split to accommodate the new node, can be improved further, as we will show in Subsection 4.8, to produce a remarkably even distribution of zone sizes.

4.5.2. Updating the routing tables. The node $\mathcal{O}$ also informs the node $\mathcal{N}$ about its 'original' neighbors in its routing table. Since the new zones of the nodes $\mathcal{N}$ and $\mathcal{O}$ are subsets of the original zone of responsibility of the node $\mathcal{O}$, their routing tables are also going to be subsets of $\mathcal{O}$'s original routing table. Hence the routing table of the new node $\mathcal{N}$ can be computed locally, after a single message exchange from node $\mathcal{O}$ that informs $\mathcal{N}$ of the contents of the original routing table of $\mathcal{O}$. Nodes in the preceding level $l-1$ that consider $\mathcal{O}$ as a neighbor (referred to as 'predecessors'), should be informed of this split.[‡‡] In the next section, we define this set of 'predecessors' formally and show through analysis that the complexity of informing the members of this set about the newly joining node is $O(\log n)$.

4.6. Self-stabilization on node departures

Nodes can depart a Ulysses network at will at any point in time. Ideally, a departing node should do so gracefully, updating the routing tables of its predecessors and handing over the keys that it holds before departing. But the Ulysses network can robustly handle ungraceful departures too, where nodes stop communicating abruptly and drop out without performing any housekeeping operations.

4.6.1. Graceful departures. When a node with identifier (P, l) leaves the network, it needs to explicitly hand over its keys, to another node at the same level. If its buddy has not been split, the two nodes must have zones of equal size. Then they should be merged to create a zone of twice this size and taken over by the buddy.

There is a small probability that, due to joins of new nodes in the meantime, the zone of departing node's

[‡‡]It is worth noting at this point that the neighbor relation in a Ulysses butterfly is not symmetric.

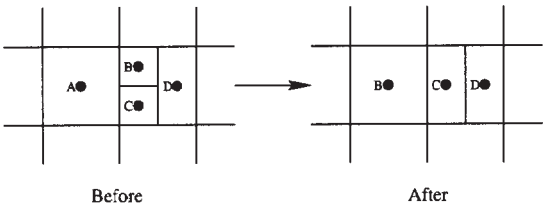

Figure 4. When buddy of a departing node (A) is split into multiple zones: the node with smallest zone (B) is *promoted* to take over departing node's zone and C merges with B's zone.

original buddy has been split into multiple zones, as shown in Figure 4. In this case, the departing node searches for a smallest node whose zone is a subset of its buddy zone. There are at least a pair of such nodes who are buddies of each other. One of these smallest nodes will be 'promoted' to take over the departing node's zone and its buddy will take over its zone. This can be done by logically emulating a depth-first search (DFS) within the buddy zone of the departing node. Each step in this logical DFS is a query for a key, issued by the departing node, and held by one of the smaller nodes in the buddy zone.

Similar to the case of join, the nodes in the previous level that consider the departing node as a neighbor should also be informed of the departure. This can again be done through explicit updates with a message complexity of $O(\log n)$ as discussed next in Subsection 4.7. The newly 'promoted' node should do the same thing about its own graceful departure from its original zone, but this time the smallest node is guaranteed to have an unsplit buddy, thus ruling out the possibility of any further promotions.

4.6.2. Ungraceful departures. Nodes in peer-to-peer networks might abruptly stop communicating due to a host of reasons, ranging from connectivity loss or power failures to large catastrophic events that might affect a significant part of the internet. We classify as ungraceful all departures that fail to complete the housekeeping operations required for a graceful departure. Handling an ungraceful departure consists of two phases—a detection phase and a repair phase. The detection itself might occur either by a timeout of periodic keep-alive messages or through an asynchronous mechanism where a node detects that its successor has failed when a query forwarding operation to that successor is unsuccessful. Timeout based detection has higher overheads due to the periodic message exchange and does not seem to offer any clear advantage over the low overhead asynchronous detection in the context of peer-to-peer networks. We thus feel that asynchronous detection is the more suitable candidate for detection of ungraceful departures in Ulysses.

Once a node has detected the ungraceful departure of one of its neighbors, it initiates and carries through all the housekeeping operations on behalf of the ungracefully departed node. The housekeeping operations themselves are the same as in the case of graceful departure. Note that the keys held by the ungracefully departed node itself cannot be recovered unless there is some kind of replication or redundancy built into the storage scheme. Such schemes are crucial to the operation of a DHT based peer-to-peer network that expects to accommodate ungraceful departures, but are orthogonal to the design objectives of Ulysses and are thus not addressed in this paper.

4.7. Cost of self-stabilization

The self-stabilization cost in Ulysses is $O(\log n)$ per join/leave, while in Chord it is $O(\log^2 n)$. The constant factor in $O(\log n)$ is about three for a join or about six for a departure. The intuitive reason for this improvement is as follows. In both Chord and Ulysses, a node appears in the routing tables of about $\log n$ other nodes. All such nodes need to be notified in the event of a join or a departure. In Ulysses, because the 'coordinates' of these neighbors are very close to the affected nodes, a property of the butterfly topology, each 'notification' only travels three hops. In Chord, on the other hand, each notification will travel $O(\log n)$ hops. This results in the difference in message complexity of self-stabilization between Ulysses and Chord, as a consequence of the Little's Law [10].

4.8. Heuristics to make the zones of responsibility similar in size

The size of routing table of a Ulysses node is proportional to the size (volume) of the portion of the DHT held by it. Thus there is a two-fold motivation for making such zones of responsibility uniform in size: (a) the load of storing DHT entries and answering queries is balanced across the network and (b) the routing tables become more uniform in size.

We designed and evaluated two optimization heuristics that augment the default join procedure described in Subsection 4.5.1 to achieve more uniformity in sizes of zones of responsibility. These two heuristics differ in handling the join request issued by a new node. The first approach, largest neighbor, checks for existence of a larger zone among the neighbors of the destination node. The second approach, largest neighbor on path, examines all neighbors of zones along the path traversed by the join request. The join is finally committed at the largest node discovered in both approaches. Neither approach results in any increase

in the cost of storage or communication over the default join procedure discussed earlier. The distribution of zone-sizes, in a simulated network of 4 million nodes, after 4 million random arrivals and departures, is shown in Figure 5. Both heuristics show a significant improvement over the plain vanilla join algorithm. Using the largest neighbor on path heuristic, over 95% of the nodes have size exactly the same as the expected size, while the remaining 5% nodes have zones that are twice or half this size, with high probability. This translates into routing tables of size around $\log_2 n$ for more than 95% of the nodes, while the routing table sizes of rest of the nodes are within the interval $[(1/2)\log_2 n, 2\log_2 n]$, with high probability. The largest neighbor heuristic performs slightly worse as it examines fewer nodes before selecting one to join. Due to its superior performance, we adopt the largest neighbor on path heuristic in Ulysses.

4.9. Choice of number of levels

One of the design parameters in the Ulysses network is the number of levels k. The total number of nodes n in the network and number of levels k determine the average degree (routing table size (r)) of nodes in the network, according to the equation: $n = kr^k$. On the other hand, the diameter of Ulysses is exactly $k + 1$. Therefore, the different tradeoff points between the size of the routing tables and the diameter of the network can be obtained by varying k. For fair comparison with other DHT-based protocols, we configure k such that the routing table size of Ulysses is about $\log_2 n$. Given a network size n, we define $\mathcal{K}(n)$ as the smallest value of k, for which the average routing table size does not exceed $\log_2 n$, i.e. $n \leqslant k(\log_2 n)^k$. In other words, given the routing table size of $\log_2 n$, setting k to $\mathcal{K}(n)$ allows Ulysses to achieve the minimum diameter. The values of $\mathcal{K}(n)$ for various ranges of n are as follows:

$$\mathcal{K}(n) = \begin{cases} 2 & n \leqslant 2^6 \\ 3 & 2^6 < n \leqslant 2^{12} \\ 4 & 2^{12} < n \leqslant 2^{18} \\ 5 & 2^{18} < n \leqslant 2^{24} \end{cases}$$

Ideally the number of levels should be $\mathcal{K}(n)$ when the network size is n. But Ulysses does not adapt k to the changing values of n dynamically. Instead, a value for k is chosen *a priori*, based on off-line information regarding the approximate size ranges of the peer-to-peer network. Fixing k in this manner can be justified as follows: First, a given value of k can be optimal for a wide range of values of n (e.g. $\mathcal{K}(n) = 5$ for $n = 256K$ to $n = 16M$). Second, even if n changes drastically so that the current number of levels is not optimal, the performance of Ulysses in terms of diameter or latency will not be affected, and only the average routing table size will increase or decrease slightly. Thus, the performance of Ulysses is quite insensitive to the choice of number of levels and even non-optimal choices do not impact performance significantly.[§§] For our experiments reported in the next section, we select the number of levels of Ulysses network to be $\mathcal{K}(n)$.

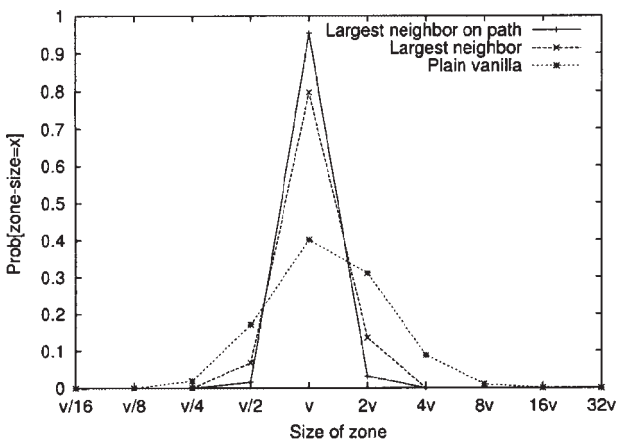

Figure 5. Distribution of zone-sizes. While only 40% nodes have zones of exactly the average size using the plain-vanilla join algorithm, the heuristics succeed in raising this fraction to 95%.

5. CORRECTNESS PROOFS OF ULYSSES PROTOCOLS

In this section, we formally specify the protocols for routing queries and handling joins/leaves in Ulysses, prove their correctness using techniques from distributed computing, and analyze their asymptotic message complexity. It consists of the following steps. First, we define the terms and state the key topology invariants that Ulysses maintains. We show that under these invariants any query will terminate in a correct fashion, i.e. routed to the correct destination. This termination statement is proved by defining a notion of progress and showing that the protocol guarantees such progress under these topology invariants. Second, we formally characterize the level of robustness that Ulysses routing achieves. Finally, we show that when a join or departure temporarily compromises these invariants the self-stabilization mechanism of Ulysses will restore them locally. We also show that the complexity

[§§]Fixing k corresponds to the tradeoff of constant diameter $(k + 1)$ and a routing table size of $n^{1/k}$. The choice of $k = \log n/\log\log n$ causes the routing table size to become $n^{\log\log n/\log n} = \log n$.

of such 'repair' is $O(\log n)$, as compared to $O(\log^2 n)$ in Chord.

We introduce some definitions and notation that will be used in later expositions. In a k level Ulysses network, for a node (P, l) where $P = a_0 a_1 \ldots a_l$, $AP(P, \{i\})$, is defined as the string formed by concatenating all a_j such that $j \in \{i, i+k, i+2k \ldots\}$, the arithmetic progression starting at i with common difference k. As explained before, $AP(P, \{i\})$ is also the coordinate of the node (P, l) in the ith dimension. The AP function can be similarly defined on a search key mapped to a tuple (α, l). In a more general form, the function $AP(\alpha, S)$, where α is any binary string and $S = \{i_1, i_2, \ldots, i_q\}$ is a set of indices, can be defined as the string formed by taking those bits in α that appear at locations j, such that j is in one of the arithmetic progressions $\{i_1, i_1 + k, i_1 + 2k \ldots\}$, $\{i_2, i_2 + k, i_2 + 2k \ldots\} \ldots$, $\{i_q, i_q + k, i_q + 2k \ldots\}$. The set S is a subset of Ω which is the set of dimensions $\{0, 1, \ldots, k-1\}$ in a k level Ulysses network. The function $AP(\alpha, S)$ retains the same order of bits as in the string α. For example, if $\alpha = a_0 a_1 \ldots a_{12}$ and $k = 5$, then $AP(\alpha, \{0, 2, 3\}) = a_0 a_2 a_3 a_5 a_7 a_8 a_{10} a_{12}$. It is not hard to verify that $AP(\alpha, \Omega) = \alpha$. We define $AP(\alpha, \emptyset)$ as the empty set $\emptyset$ by convention. We define merge(α, S, β) as a 'hybrid' of two keys $\alpha = a_0 a_1 \ldots a_{m-1}$ and $\beta = b_0 b_1 \ldots b_{m-1}$ as follows. It takes i'th coordinate from α if $i \in S$ and from β otherwise. For example, when $m = 13$ and $k = 5$, merge$(\alpha, \{0, 2, 3\}, \beta)$ is equal to $a_0 b_1 a_2 a_3 b_4 a_5 b_6 a_7 a_8 b_9 a_{10} b_{11} a_{12}$. With this definition of the function $AP(.)$, we can define the links of the Ulysses network: a node (P, l) has a link to the node $(P', l+1)$ if and only if $AP(P, \overline{\{l+1\}}) \subseteq AP(P', \overline{\{l+1\}})$ or $AP(P', \overline{\{l+1\}}) \subseteq AP(P, \overline{\{l+1\}})$. Here, $\overline{\{l+1\}} = \Omega - \{l+1\} = \{0, 1, \ldots, l, l+2, \ldots k-1\}$.

The correctness statements of Ulysses can be summarized as follows:

(1) *Zoning invariant*: Each level cuboid is partitioned into disjoint zones (prefixes). So for each key (α, i), there is a unique zone that handles it. We denote this zone as $R(\alpha, i)$.
(2) *Connectivity invariant*: We let $N(P, i)$ denote the set of neighbors of a zone (P, i) at the level $i+1$, and let $H(P, i, j)$ denote its set of shortcut neighbors at level $j, j \neq i, i+1$. The connectivity invariants are, for all (P, i) in the network, (1) $N(P, i) = \{R(\alpha, i+1) | AP(\alpha, \overline{\{i+1\}}) \in AP(P, \overline{\{i+1\}})\}$ for normal neighbors and (2) $H(P, i, j) = \{R(\alpha, j) | \alpha \in P\}$, $0 \leqslant j \leqslant k-1, j \neq i, i+1$. Here we simply formalize our intuitive characterization of Ulysses' connectivity in Subsection 4.2 and Figure 2.
(3) *Progress statement*: Assume that both the zoning and the connectivity invariants hold. Then our routing algorithm (shown in Figure 3) can always progress: it can always find a 'next-hop' unless the termination condition has already been met, and this 'next-hop' will lock the row of the query key (α, i) at the level that the 'next-hop' lies in. This statement is formalized and proved in Theorem 1.
(4) *Termination statement*: Assume that both the zoning and the connectivity invariants hold. Any key (α, i) will be eventually routed to its destination in no more than $k+1$ steps. This statement is formalized and proved in Theorem 2.
(5) *Self-stabilization statement*: The self-stabilization mechanisms of Ulysses can restore the zoning and connectivity invariants when a join and departure (graceful or ungraceful) operation compromises them. The message complexity of the repair is $O(\log n)$. This is shown in Subsection 5.3.
(6) *Robustness statement*: Even when a large percentage of the nodes in the network is faulty (say 20%), the queries can still be routed to the non-faulty destination using our 'randomized detour' algorithm, also shown in Figure 3, with high probability. This property is formally stated in Theorem 3.

5.1. Progress and termination

We first introduce a technical lemma that will be used in several later proofs. It states that if a key α is locked by a zone P at two sets of levels S_1 and S_2, then it is locked by P at the set of levels $S_1 \bigcup S_2$.

Lemma 1. If $AP(\alpha, S_1) \in AP(P, S_1)$ *and* $AP(\alpha, S_2) \in AP(P, S_2)$ *then* $AP(\alpha, S_1 \bigcup S_2) \in AP(P, S_1 \bigcup S_2)$

Proof. Since $AP(\alpha, S_1) \in AP(P, S_1)$, by the definition of AP, there exists $\beta_1 \in P$ such that $AP(\beta_1, S_1) = AP(\alpha, S_1)$. Since $AP(\alpha, S_2) \in AP(P, S_2)$, there exists $\beta_2 \in P$ such that $AP(\beta_2, S_2) = AP(\alpha, S_2)$. Then let $\gamma = \text{merge}(\beta_1, S_1, \beta_2)$. It is not hard to verify that $\gamma \in P$ and $AP(\gamma, S_1 \bigcup S_2) = AP(\alpha, S_1 \bigcup S_2)$. Therefore $AP(\alpha, S_1 \bigcup S_2) \in AP(P, S_1 \bigcup S_2)$. ■

Suppose that a query for (α, j) reaches a node (P, i). We prove that the routing of the query at the current node P satisfies the progress statement above. We denote as $L(\alpha, P)$ the set of dimensions that P has 'locked' on the destination α, i.e. $L(\alpha, P) = \{i | AP(\alpha, i) \in AP(P, i), i \in \Omega\}$. We will show that (1) the appropriate next-hop node P' always exists and (2) upon the delivery of the query to P', one additional dimension ($(i+1)^{\text{th}}$) gets locked.

Theorem 1—Progress Statement. *Suppose* $\alpha \notin P$ *and* $L(\alpha, P) \neq \Omega$, *the message will be routed from* P *at level* i *to some* P' *at the level* $i+1$ *such that* $AP(\alpha, L(\alpha, P')) \supseteq AP(\alpha, L(\alpha, P)) \bigcup \{i+1\}$.

Proof. The existence of such a P' is from a more general lemma below. Note that the algorithm will route the packet to one of such P' (line 9 in Figure 3) at the level $i+1$. ■

Lemma 2. *Suppose* $AP(\alpha, S) \in AP(P, S)$. *Then* (P, i) *must have a link to some* $(P', i+1)$ *such that* $AP(\alpha, S \bigcup \{i+1\}) \in AP(P', S \bigcup \{i+1\})$.

Proof. Choose an arbitrary point $\beta \in P$. Let $\gamma = \text{merge}(\alpha, S \bigcup \{i+1\}, \beta)$. From the definition of merge and the assumption, we know that $AP(\gamma, S) = AP(\alpha, S) \in AP(P, S)$. Also, we know that $AP(\gamma, \overline{S \bigcup \{i+1\}}) = AP(\beta, \overline{S \bigcup \{i+1\}}) \in AP(P, \overline{S \bigcup \{i+1\}})$. By Lemma 1, $AP(\gamma, \overline{\{i+1\}}) \in AP(P, \overline{\{i+1\}})$. Therefore, by the connectivity invariant, $R(\gamma, \{i+1\}) \in N(P, i)$. Now we find such a $(P', i+1)$, which is $R(\gamma, \{i+1\})$. ■

This theorem leads to the following corollary stating that after l forwarding operations, $l \leqslant k$, at least l dimensions have been locked.

Corollary 1. *If a query for key* α *has been forwarded* l *times* $(l \leqslant k)$ *and reaches a block* P, *then* $|L(\alpha, P)| \geqslant l$.

From the above proposition and corollaries, it can be concluded that the routing algorithm correctly routes a query for (α, j) originating at any random node to some node (D, i), such that $\alpha \in D$. From here, the query is forwarded to the destination through a shortcut link. Thus the routing algorithm correctly routes any query to its destination in at most $k+1$ steps. This intuition is captured in the following theorem and proof.

Theorem 2—Termination Statement. *A query will be routed to the destination after no more than* $k+1$ *statements.*

Proof. If the query has been routed to the destination within k steps, then we are done. Otherwise, suppose the query reaches a node P which is not the destination, in k steps. Then we know that $L(\alpha, P) = \Omega$ since $|L(\alpha, P)| = k$ by Corollary 1. So $\alpha = AP(\alpha, L(\alpha, P)) \in AP(P, L(\alpha, P)) = P$. So all coordinates have been locked and the only the level is different. Then according to our routing algorithm (line 8 in Figure 3), the query will follow a shortcut link to the destination, the existence of which is due to the connectivity invariant. ■

5.2. Robustness

Our randomized detour algorithm minimizes the penalty of detour. After the 'detour', at most zero or one (when 'vicious cycle avoiding' is needed) previously locked dimensions become unlocked in order to route around the problem. If faults are not encountered after the detour, then the progress and termination statements above guarantee that the query reaches the destination in at most two to three additional hops. The following theorem captures this intuition. We omit its proof since it is quite straightforward from our description in Subsection 4.4.

Theorem 3—Robustness Statement. *Suppose a query for key* α *reaches a node* P. *Suppose the next hop* P', *as specified in Theorem 1, is faulty at the moment and* P' *is not the destination. Then the following are true: (1) if* $L(\alpha, P) = \Omega - \{i+1\}$, *then after two randomized steps as specified in the routing algorithm (Subsection 4.4), it will land on a* P'' *such that* $L(\alpha, P'') = \Omega - \{i+1, i+2\}$, *thereby avoiding the vicious cycle as discussed in Subsection 4.4 (2) Otherwise, after one randomized step, it will land on a node* P'' *such that* $L(\alpha, P'') = L(\alpha, P)$.

5.3. Self-stabilization and its message complexity

Theorem 4—Self-Stabilization Statement. *Suppose the connectivity invariants concerning the links that point to or from a region* $P \bigcup buddy(P)$, *and the zoning invariant concerning* $P \bigcup buddy(P)$ *are satisfied, before a join/departure occurs (to P). Then after the execution of the join and leave protocol as specified in Subsections 4.5 and 4.6, the aforementioned invariants remain satisfied for the region* $P \bigcup buddy(P)$.

A join or departure will temporarily compromise the aforementioned topology invariants at certain places. As stated in the above theorem, our self-stabilization mechanism will repair such 'faults' and restore these invariants. Like in other DHT schemes, Ulysses can recover from multiple concurrent faults simultaneously due to its distributed nature.

Additionally Ulysses has a salient feature that the complexity of the repair, in terms of messages transmitted between DHT nodes, are $O(\log n)$ per fault, compared to $O(\log^2 n)$ per fault in Chord. In Ulysses, the constant factor for this $O(\log n)$ is approximately three in the case of join and six in the case of departure. The reason we can achieve this remarkable saving, as mentioned before, is the following. In Ulysses, same as in Chord and other DHT schemes, $O(\log n)$ predecessor nodes need to be notified about this join or departure. However, each such notification travels

about $O(\log n)$ overlay hops in Chord, but only travels three overlay hops in Ulysses, as described below.

5.3.1. Join. A new node joining the network needs to obtain its own routing table and notify its predecessors so that the connectivity invariant is maintained. Without loss of generality, assume that the new node joins a node (Q, i) in the network. One message exchange is needed between the new and old nodes to 'split' the routing table. Also, the old node should notify the set of nodes in level $i-1$, that maintains a link to it, about the split. This set is equal to $\{R(\alpha, i-1) \mid AP(\alpha, \overline{\{i\}}) \in AP(Q, \overline{\{i\}})\}$, which we denote as $\text{Pred}(Q, i)$. The expected size of $\text{Pred}(Q, i)$ is about $\log n$ by the symmetry of butterfly topology. Also, any predecessor of (Q, i) overlaps with it in all but one dimension. Formally, $\forall(Q', i-1) \in \text{Pred}(Q, i)$, either $AP(Q', \Omega - \{i\})$ is a substring of $AP(Q, i)$ or *vice versa*. Since the members of the set $\text{Pred}(Q, i)$ have identifiers that differ from Q only in the bits that belong to $AP(Q, i)$, the notification being delivered needs to be locked only in the dimension i. Using the optimized version of our routing algorithm described in Subsection 4.4.2, exactly three hops—one to lock the query in dimension i, and two over shortcut links to get to the correct level before and after locking the query in this dimension—are needed to deliver a notification to a predecessor.

5.3.2. Departure. The only difference between the graceful and ungraceful departures is that in the former case, the node initiating the updates is the departing node itself, while in the latter case it is one of the predecessors that has asynchronously detected the ungraceful departure. In both cases, the set of nodes that need to be notified is exactly the same: the set of nodes that have a link to the nodes affected by the departure. It can be shown that the average number of notifications is about $2\log n$, since departure usually results in a larger region than join. The expected number of hops that each notification travels is 3, for the same reason explained above.

6. PERFORMANCE EVALUATION

We have carefully evaluated various aspects of Ulysses performance using simulation. The findings of our simulation study can be summarized as follows:

- In Subsection 6.1, we show that the worse-case and average path lengths of routes taken by queries, are far less for Ulysses than for other DHT-based protocols. This reduction is achieved while keeping the routing table size similar to that of the other protocols.
- In Subsection 6.2, we show that given the same offered query load and network size, the average query traffic going through an overlay link or node is much lower for Ulysses than for other protocols, as expected from the Little's law. The same is true for nodes and links in the underlying physical network.
- In Subsection 6.3, we show that Ulysses is very robust. Even when a large percentage (e.g. 20%) of nodes become faulty, most (99.95%) of the queries destined for non-faulty nodes can still reach the destination, along a path only slightly longer than in fault-free routing.

We choose Chord [1] as the benchmark for our experiments, since Chord represents the whole family of DHT based protocols that have a network diameter of $O(\log n)$ and routing table size of $O(\log n)$. Among others, this family includes Pastry [3] and Tapestry [2] which use a base of 2^b instead of 2, hence achieving a diameter of $\log_{2^b} n = \log_2 n/b$, but with a larger routing table of size $(2^b/b)\log_2 n$. Ulysses can also attain similar gain of smaller diameter and larger routing table by using a larger base. However, for objectivity of evaluation, we work with base 2 only keeping in mind that our results can be extended to larger bases as well. Therefore, we compare Ulysses against Chord with both protocols having the same routing table size of $\log n$. Similarly, we do not consider the class of protocols (e.g. CAN [5]) that achieve a network diameter of $O(dn^{1/d})$ with a routing table of size $O(d)$ because these protocols achieve their optimum diameter at $d = \log n$, which reduces them to the same family as Chord. Since the primary design objective[⊥] of Ulysses is to achieve a low diameter, such protocols would show poorly against Ulysses for other choices of d.

We implemented an event-driven simulator for evaluating different DHT-based protocols. It simulates node arrivals and departures as well routing queries. It is written in C/C++ and executes as a single process. The number of nodes in the peer-to-peer network is varied from 256 to 4 000 000. Our implementation of the Chord protocol is from the description given in Reference [1]. The following sections describe our experiments and analysis of the results.

6.1. Query path length

Our simulation results verify our expectation from the theoretical analysis that Ulysses achieves significantly lower

[⊥]Proposed enhancements to protocols like CAN [5], Pastry [14] etc. take network proximity into account during node joins to construct overlays that have a good correspondence between the topology of the overlay and that of the underlying network. Similar enhancements could be proposed for Ulysses too, but we focus on achieving the optimum tradeoff in the overlay, and leave the exploration of such enhancements to future work.

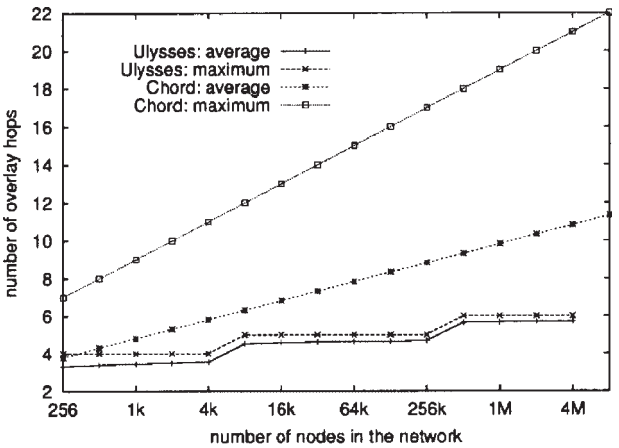

Figure 6. The average and maximum number of hops required for a query to be routed correctly to its destination in an overlay network with n nodes.

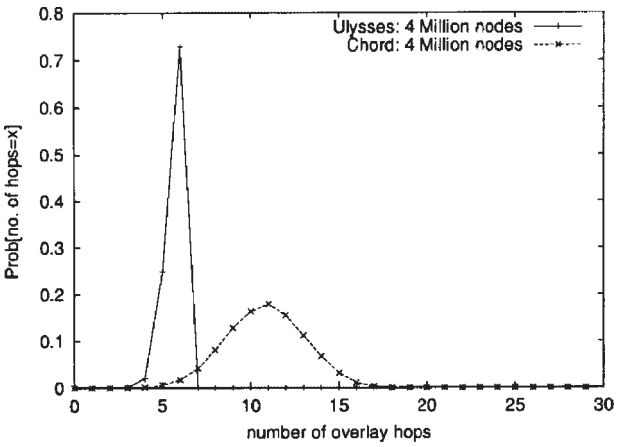

Figure 7. The probability density function of number of hops required for a query to be routed correctly to its destination in an overlay network with 2^{22} nodes.

worst-case and average query latency than Chord. The query path latency is measured in terms of overlay network hops. We assume a simple model for the offered query load: queries are generated randomly and uniformly at each node, destined for keys that are uniformly distributed in the key-space. As discussed in Subsection 4.9, we tune the number of levels k such that the average routing table size is below $\log n$, where n is total number of nodes.

We have simulated the worst-case and average path lengths for both Ulysses and Chord, as a function of the number of nodes n (x-axis, in the log scale). The results are plotted in Figure 6. Two curves are plotted for both Ulysses and Chord, denoting the worst-case and the average query path lengths measured in the number of overlay hops respectively. The 'steps' on both Ulysses curves are due to the increase of parameter k when n becomes larger, as discussed in Subsection 4.9. As expected, curves for Chord coincide with functions $\log_2 n$ for the worst-case and $1/2 \log_2 n$ for the average case. They look 'linear' in the figure since the x-axis is in log scale. We can see from Figure 6 that both the average and maximum number of hops required by Ulysses are less than the average number of hops required by Chord. The difference is more pronounced at a larger network size. These differences highlight the better scaling properties[#] of Ulysses compared to Chord. Figure 7 shows the probability distribution function of overlay path lengths in Ulysses and Chord for a network size of 2^{22} (4 million) nodes. We can see from the figure that the mean number of overlay hops for Ulysses is less than that of Chord. The sharp peak of the Ulysses curve, and hence its low variance, indicate that the length of most of its paths is very close to the mean.

6.2. Reduction in traffic load

By Little's law, given the same offered load of DHT queries and same network size, Ulysses is expected to generate much less overall network traffic than Chord, since each query travels (i.e. 'stays in the network') for less number of overlay hops. In this subsection, we demonstrate this advantage through simulations. As a consequence, the amount of traffic that goes through each link and node also goes down proportionally.

We define the traffic intensity as the number of queries that traverse a link or node in the underlying physical topology. A network topology comprising of 10 320 routers with an average degree of 3.92 is generated using the Transit-Stub graph model from the GT-ITM topology generator [16]. We attach a set of end-hosts to each of the stub routers. The number of end-hosts attached to a stub router is chosen from a normal distribution with mean 14.0 and standard deviation 4.0. The resulting graph has approximately 141 000 end-hosts. In each experiment, we uniformly and randomly pick end-hosts of the underlying network topology to construct the overlay network. Thus a single hop in the overlay network corresponds to a path of routers between end-hosts in the underlying

[#]Since neither Ulysses nor Chord optimizes the mapping of the overlay network to the underlying physical topology, we observed similar reduction in the query path length when measured in underlying network link latencies [15].

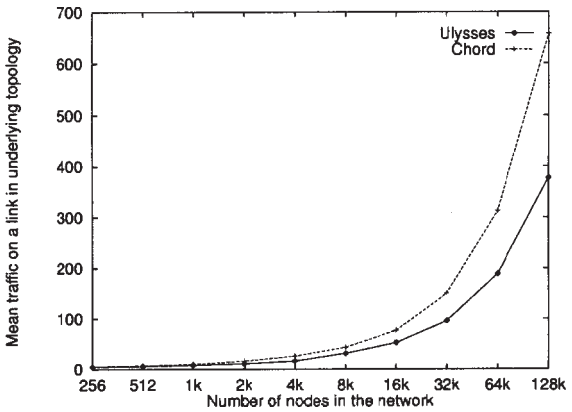

Figure 8. Variation of traffic density with different sizes of overlay networks.

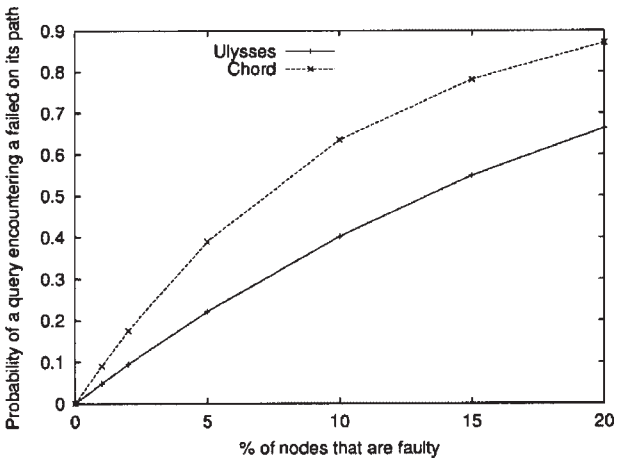

Figure 9. Probability that a query encounters a node failure as we vary the percentage of node failures in a 4 million node overlay network.

topology. The total number of queries simulated in each run is equal to the size of the overlay network. We generate one query from each overlay node to a random destination uniformly drawn from the key space. Therefore, the offered load increases linearly with the network size. Figure 8 plots the average traffic intensity on links in the underlying physical topology, as a function of the size of the overlay network. We can see that, as the network size increases and the offered load increases in the same proportion, the traffic intensity increases for both Chord and Ulysses. However, both the absolute value of traffic intensity and its rate of increase are higher for Chord than for Ulysses. This confirms our expectation since the path length grows faster in Chord than in Ulysses with increasing network size.

6.3. Robustness

In this section, we demonstrate the robustness of Ulysses through extensive simulation study. We evaluate the success rate for a query to detour around node failures (due to ungraceful departures) and the penalty, in terms of additional overlay hops in path length, for such detours. The robustness of Ulysses is evaluated in comparison with Chord by simulating sudden ungraceful departures of a fraction of the node population (e.g. due to a massive earthquake or power outage). The fraction of nodes that are 'switched off' are chosen randomly and uniformly from an overlay network of 4 million nodes. The percentage of nodes that fail is varied from 1 to 20%. A simple query traffic model with random source and destination as described in Subsection 5.1 is simulated here. The probabilities reported in these plots on robustness are computed from the results of simulating 10^5 queries in each case.

Figure 9 plots the probability that a query encounters a node failure while following a path to its destination using the forwarding algorithm of corresponding protocol. For the same percentage of node failures, the probability that a query in a Chord network encounters a failure on its path is higher than in Ulysses network because of the longer paths (hence, more nodes visited per query) in Chord. Although encountering a node failure on the path taken by a query does not impact the reachability to its final destination as we can route around the node failure, this phenomenon can have ramifications that affect other 'in-band' mechanisms, like asynchronous detection of failures etc.

Routing a query around a node failure will inevitably increase the path length for routing. For Chord, routing around a failure involves forwarding the query to a node preceding the failed node in the finger table, thus adding one extra hop to the path of the query. In Ulysses, as shown in Theorem 3, the penalty incurred is usually two extra hops, but can be up to three in the worst case of 'vicious cycle' as described in Subsection 4.4. Figure 10 plots the average number of hops required to route a query to its destination against the percentage of failed nodes. Both Chord and Ulysses show an increase in the average number of hops per query with increasing percentage of failed nodes. We noted that this increase is sharper for Ulysses than for Chord in spite of the higher probability of queries in the Chord network encountering a node failure on their

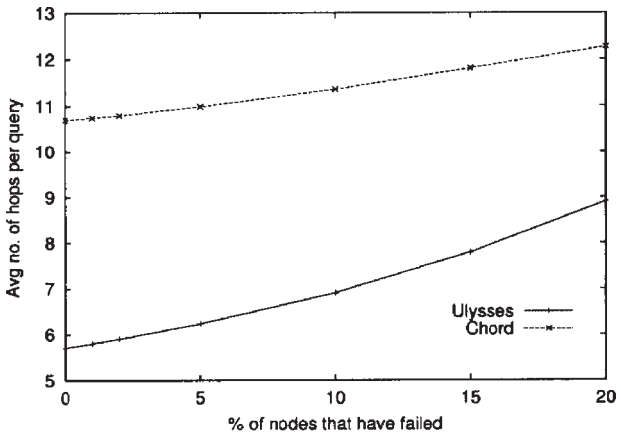

Figure 10. Average number of hops required to route a query versus percentage of failed nodes in a 4 million node network.

path. This is due to the aforementioned higher penalty for routing around failures in a Ulysses network. However, even with its slightly higher slope, the curve for Ulysses lies below that for Chord throughout the range (1–20) of node failures that was simulated.

7. OTHER RELATED WORK

The Viceroy network [12], also based on the butterfly topology, has been proposed to achieve $O(\log_2 n)$ network diameter with constant routing table size. Viceroy achieves this by mapping a butterfly topology onto a basic Chord ring [1] with only the successor and predecessor links. Routing in Viceroy consists of taking the butterfly links in two phases to reach within a distance of $O(\log n)/n$ from the destination and then taking the successor or predecessor links of the Chord ring to reach the destination in the third phase. The minimum diameter of Viceroy is $3\log_2 n$ and is $O(\log_2 n)$ with high probability. However, the random distribution of nodes on the Chord ring imply that the worst case diameter can be much larger. For similar reasons, the worst case stress at nodes and edges is $O(\log_2 n)$ times the average in Viceroy while it is only a constant times the average in Ulysses.

Koorde, proposed in a contemporaneous work [17], uses the de Bruijn graph to achieve a diameter of $O(\log n/\log\log n)$ with $O(\log n)$ neighbors per node. However this bound is achieved by Koorde only in the expectation and with a larger constant in the 'big O' notation. It is also not clear whether Koorde can self-stabilize like Chord or Ulysses. It seems that Koorde relies on an underlying Chord substrate for self-stabilization. Koorde and Ulysses can be seen as two parallel ways of achieving the degree-diameter tradeoff, with Koorde achieving a deterministic bound on the number of neighbors and a probabilistic bound on the diameter while Ulysses achieves a deterministic bound on the diameter with a probabilistic bound on the routing table size.

Naor and Wieder [18] propose a similar mechanism to use the de Bruijn graph to achieve a diameter of $O(\log n/\text{loglog}\, n)$ with $O(\log n)$ neighbors per node. Like Koorde, this mechanism too achieves only a probabilistic bound on the diameter. Further, they require cooperation among $\log n$ nodes for storing a key. Since keys stored in the DHT are more numerous than nodes, this requires a large amount of state at nodes. Both Koorde and Naor require the number of neighbors to be $\log n$. This requires an *a priori* knowledge of the logarithm of network size (n). Ulysses on the other hand requires a knowledge of '$\log n/\log\log n$' which is much less sensitive to changes in network size.

In Reference [19], the authors present ODRI that leverages on the same theoretical properties of de Bruijn graphs to achieve the optimal degree-diameter tradeoff with high probability. An analysis of graph-theoretical properties of de Bruijn graphs that makes them more suitable for use in peer-to-peer networks, is also provided. However, Reference [19] does not provide operational details of ODRI such as its mechanisms for self-stabilization and robust routing, making a more detailed comparison with Ulysses impossible. A result in Reference [19] showing that the butterfly topology has larger diameter than the de Bruijn network does not hold for Ulysses due to our introduction of shortcut links in Subsection 3.2.

8. CONCLUDING REMARKS

We have presented the design and analysis of Ulysses, a peer-to-peer protocol that meets the theoretical lower bounds for the tradeoff between routing table size and network diameter. In addition to reducing the diameter (i.e. worst cast query routing length), Ulysses also reduces the average query routing length as compared to other protocols with similar routing table sizes. The reduction in query routing length implies a number of additional advantages, including reduced traffic at nodes and links. Ulysses achieves the minimum diameter deterministically, compared to related efforts that achieve this lower bound with high probability.

Ulysses is based on a butterfly topology with significant adaptations to achieve the aforementioned properties. In brief, Ulysses includes shortcut links to remove stress on certain edges, a novel method for assigning peers to locations in the butterfly and for assigning portions of the key space to peers, and a collection of self-stabilization and robustness techniques to allow efficient and correct operation under network dynamics. With an increasing number of proposals to build peer-to-peer applications and services over the routing primitives provided by DHT-based networks, Ulysses offers a degree-optimal solution implementing these primitives.

REFERENCES

1. Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM'01*, 2001.
2. Zhao BY, Kubiatowicz J, Joseph A. Tapestry: an infrastructure for fault-tolerant wide-area location and routing. Technical Report, U.C. Berkeley Technical Report UCB/CSD-01-1141, 2001.
3. Rowstron A, Druschel P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
4. Plaxton CG, Rajaraman R, Richa AW. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of ACM Symposium on Parallel Algorithms and Architectures*, 1997.
5. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM'01*, 2001.
6. Stoica I, Adkins D, Zhaung S, Shenker S, Surana S. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM'02*, 2002.
7. IRIS: Infrastructure for Resilient Internet Systems. http://www.project-iris.net
8. Ratnasamy S, Shenker S, Stoica I. Routing algorithms for DHTs: some open questions. In *Proceedings of 1st Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.
9. Xu J, Kumar A, Yu X. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE JSAC Special Issue on Recent Advances in Service Overlay Networks*, November 2003.
10. Kleinrock L. *Queueing Systems*, Vol. I and II, John Wiley and Sons: New York, 1975.
11. Siegel HJ. Interconnection networks for SIMD machines. *Computer* 1979; **12**(6):57–64.
12. Malkhi D, Naor M, Ratajczak D. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of ACM PODC*, 2002.
13. Francis P. Yoid: extending the internet multicast architecture. Unrefereed report, 38 pages, April 2000.
14. Castro M, Druschel P, Hu Y, Rowstron A. Exploiting network proximity in distributed hash tables. In *International Workshop on Future Directions in Distributed Computing*, 2002.
15. Kumar A, Merugu S, Xu J, Zegura E, Yu X. Ulysses: a robust, low-diameter, low-latency peer-to-peer network. Technical Report GIT-CC-03-30, College of Computing, Georgia Institute of Technology, 2003.
16. Zegura E, Calvert K, Donahoo MJ. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Transactions on Networking* 1997; **5**(6):770–783.
17. Kaashoek MF, Krager DR. Koorde: a simple degree-optimal distributed hash table. In *IPTPS*, February 2003.
18. Naor M, Wieder U. A simple fault tolerant distributed hash table. In *IPTPS*, February 2003.
19. Loguinov D, Kumar A, Rai V, Ganesh S. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *Proceedings of ACM SIGCOMM'03*, 2003; pp. 395–406.

AUTHORS' BIOGRAPHIES

Abhishek Kumar received the B.Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Delhi, in 2002. He is currently a Ph.D. student in Computer Science at Georgia Institute of Technology. His current research interests include applications and algorithms for peer-to-peer networks, algorithms and mechanisms for high speed traffic monitoring, and internet scale analysis of worm propagation. He received the Best Student Paper Award at the ACM SIGMETRICS/IFIP Performance joint conference in June, 2004. In the past, he has worked on mobile ad hoc networks, active networks, active queing mechanisms and on flow and congestion control in networks.

Shashidhar Merugu received the B.Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Madras, and the M.S. degree in Computer Science from Georgia Institute of Technology. He is currently a Ph.D. candidate in Computer Science at Georgia Institute of Technology. His research examines the interplay between routing messages and network topology design in two classes of networks—unstructured peer-to-peer networks and sparsely-connected ad hoc networks. His current research interests include applications and algorithms for peer-to-peer networks, overlay networks and environments for mobile and ubiquitous computing. In the past, he has worked on active networks, packet classifiers and real-time systems.

Jun (Jim) Xu is an assistant professor in the College of Computing at Georgia Institute of Technology. He received his Ph.D. in Computer and Information Science from The Ohio State University in 2000. His current research interests include data streaming algorithms for the measurement and monitoring of computer networks, network algorithms and data structures, network security, and performance modeling and simulation. He received the NSF CAREER Award in 2003 for his ongoing efforts in establishing fundamental lower bound and tradeoff results in networking. He is a co-author of a paper that received the Best Student Paper Award from 2004 ACM Sigmetrics/IFIP Performance joint conference, and the thesis advisor of the student winners.

Ellen Zegura received the B.S. degree in Computer Science (1987), the B.S. degree in Electrical Engineering (1987), M.S. degree in Computer Science (1990) and the D.Sc. in Computer Science (1993) all from Washington University, St Louis, MO. Since 1993, she has been on the faculty in the College of Computing at Georgia Tech. The theme of her research work is the development of wide-area (internet) networking services. Most of her work in this area falls into three categories: (1) measurement and modeling, (2) design of new services and (3) investigation of paradigms and platforms to support new services.

Xingxing Yu is a professor in the School of Mathematics at Georgia Institute of Technology. He received the M.S. degree in Mathematics from East China Normal University in 1986 and his Ph.D. in Mathematics from Vanderbilt University in 1990. His current research interests include graph theory, knot theory and discrete algorithms.