Ulysses: A Robust, Low-Diameter, Low-Latency Peer-to-Peer Network

Abhishek Kumar Shashidhar Merugu Jun (Jim) Xu
College of Computing,
Georgia Institute of Technology,
Atlanta, GA 30032
{akumar, merugu, jx}@cc.gatech.edu

Xingxing Yu
School of Mathematics,
Georgia Institute of Technology,
Atlanta, GA 30032
yu@math.gatech.edu

***Abstract*—**
A number of Distributed Hash Table (DHT)-based protocols have been proposed to address the issue of scalability in peer-to-peer networks. However, it remains an open question whether there exists a DHT scheme that can achieve the theoretical lower bound of $\frac{\log n}{\log\log n}$ on network diameter when the average routing table size at nodes is no more than $\log n$. In this paper, we present Ulysses, a peer-to-peer network based on the butterfly topology that matches this theoretical lower bound. 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/departures, as provided by existing DHT-based schemes. The protocol is formally verified for its correctness and robustness using techniques from distributed computing. The performance of the protocol has been evaluated using both analysis and simulation.

I. 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], [2], [3], [4], [5]). These protocols generally rely upon Distributed Hash Tables (DHTs) to allow each *node* to maintain a relatively small routing table, while achieving 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]).

An important and fundamental question in the space of DHT-based peer-to-peer protocols concerns 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. The fundamental nature of this question should be clear. It is further important for two reasons. The number of nodes n in a peer-to-peer network could be very large, hence differences in values that are insignificant for small n have the potential to be important. Second, because each hop in a peer-to-peer network is an application-layer overlay hop, a savings in query hops is magnified when one considers the network layer impact.

It was observed in [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 [1], [2], [3], [4]), or (2) a routing table of size d and network diameter of $O(dn^{1/d})$ (including [5]). It was asked in [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 the lower bound: the actual bounds are $\Omega(\frac{\log_2 n}{\log_2 \log_2 n})$ and $\Omega(\log_d n)$ respectively.

An open question posed in [9] concerns the design of a DHT-based scheme that achieves the lower bounds without causing *congestion*, where congestion 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 DHT-based protocol based on a butterfly topology that meets the theoretical lower bounds without congestion.* In particular, with a routing table size of about $\log_2 n$, Ulysses can reach the diameter of $\lceil \frac{\log_2 n}{\log_2 \log_2 n} \rceil + 1$, as compared to $\log_2 n$ in all existing schemes with similar routing table size[1]. This represents a significant reduction in network diameter. For example, in a network of 1 million ($\approx 2^{20}$) nodes, the average diameter of Chord[2] [1] is 20, but that of Ulysses is 6, with similar average routing table sizes (20 for Chord and around 17 for Ulysses). Since each link here is an application-layer hop, the search time can be reduced considerably. For larger values of n, the advantages of Ulysses become even more pronounced (e.g., 30 hops versus 7 hops when $n = 10^9$).

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 $\frac{1}{2}\log_2 \log_2 n$ 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 meeting three primary chal-

[1] As will be described later, the actual routing table size may vary between $0.5log_2 n$ and $2log_2 n$ from node to node, with an average of no more than $\log_2 n$.

[2] 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 $O(\log n)$.

lenges. First, the basic butterfly has edge congestion where some edges carry significantly more traffic than average. To address this, we add edges to the butterfly to avoid edge congestion 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, as well as 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 solution approach in Section II. After introducing the static butterfly topology and its shortcomings in Section III, we describe Ulysses in detail in Section IV. We present a verification of our design using formalism from distributed computing in Section V. In Section VI, we discuss results of our performance evaluation by simulation. Section VII has other related work followed by concluding remarks in Section VIII.

II. 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.

A. Design Objectives:

Low latency routing: Ulysses strives to achieve the lowest possible network diameter with a routing table size of about $\log n$ per node, without causing excessive "stress" at any overlay link or node. While the butterfly topology can help us achieve this low diameter in theory, it remains a very challenging task to embed the butterfly structure in a real-world 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 about $\log n$ at each node. Both properties need to be maintained despite node joins and leaves.

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.

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.

B. 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 DHT based peer to peer networks have only used the constant out-degree version of this topology that achieves a diameter of $O(\log n)$ [12]. To the best of our knowledge, 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 no more than $\log n$. After proposing the enhanced version of the butterfly topology in the next section we concentrate our design effort, described in section IV on embedding this topology in a peer-to-peer network 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 sections IV-E and IV-F. In Section V, we show that with its self-stabilization mechanisms, Ulysses will recover to a correct state. A salient feature of Ulysses' self-stabilization mechanisms is their low cost, discussed further in section V-C.

Ulysses protocol 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% percent of nodes are rendered inoperable, the system can still serve all the queries destined for non-faulty nodes with very high probability.

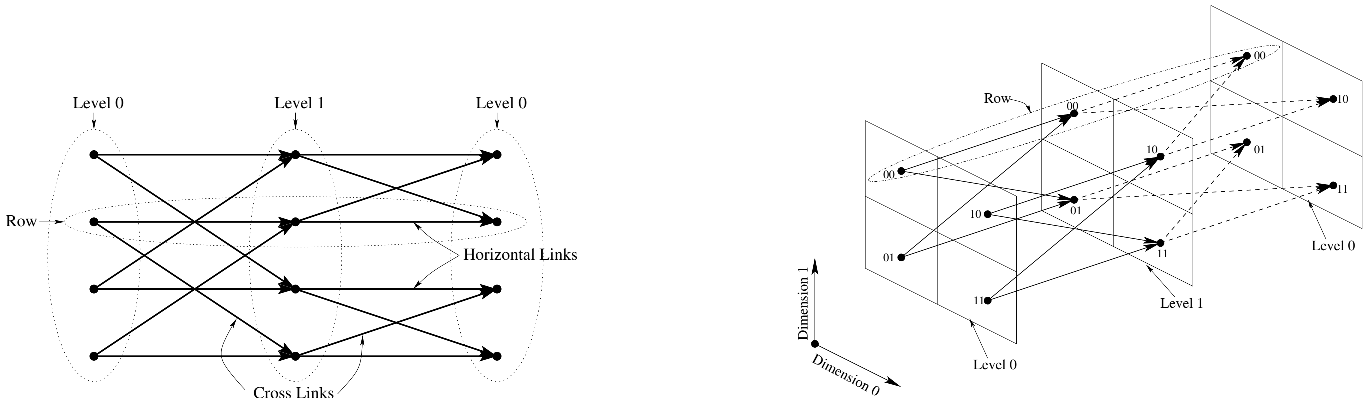
III. The static butterfly topology

The Ulysses protocol creates a network whose structure 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 this *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 then describe a solution to this problem of high *edge stress* by adding *shortcut links*. This solution is subsequently used in the Ulysses network as explained in Section IV.

A. 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 $(x_0, x_1, \cdots, x_{k-1}; i)$, where $0 \leq i \leq k-1$ and $0 \leq x_0, x_1, \cdots, x_{k-1} \leq r-1$, i.e. $x_0, x_1, \cdots, x_{k-1}$ are base r digits. For each vertex $(x_0, x_1, \cdots, x_{k-1}; i)$, we refer to i as its *level*[3] and $(x_0, x_1, ..., x_{k-1})$ as its *row*. From each vertex $(x_0, x_1, \cdots, x_{k-1}; i)$, there is a directed edge to all vertices of the form $(x_0, x_1, \cdots, x_i, y, x_{i+2}, \cdots, x_{k-1}; i+1)$ when $i \neq k-1$ and $(y, x_1, ..., x_{k-1}; 0)$ when $i = k-1$. The routing from

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



(a) Static Butterfly Topology

(b) Alternative Representation

Fig. 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 8-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.

vertex $(x_0, x_1, \cdots, x_{k-1}; i)$ to vertex $(y_0, y_1, \cdots, y_{k-1}; j)$ 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 $(y_0, y_1, ..., y_{k-1})$ to level j.

Figures 1(a) and 1(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.

In [9], the notion of *edge congestion free* is defined under the assumption of *uniform traffic load*[4] between any two nodes in the peer-to-peer network. To avoid confusion with the usage of the term congestion in the context of transport layer congestion control, we shall use the term *edge stress free* instead of *edge congestion free*.

Definition 1: We say that a network is *edge stress free* if the amount of traffic going through any edge in the network is no more than c times the average. Here $c \geq 1$ is a small constant.

The static (k, r) butterfly is not *edge-stress free*. Consider the edges going from a node $(x_0, x_1, \cdots, x_{k-1}; i)$ to $(x_0, x_1, \cdots, x_{k-1}; i + 1)$, i.e. the edges between nodes with 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 $\frac{k-1}{2}$ horizontal links in the second phase of routing as described above. Therefore, a horizontal link carries about $\frac{r}{2}$ times as much traffic as a cross link.

[4]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*.

B. Shortcut links to remove stress

We solve this stress problem by adding $k - 2$ "shortcut" links from the node $(x_0, x_1, \cdots, x_{k-1}; i)$ at level i to nodes $(x_0, x_1, \cdots, x_{k-1}; j)$ at level j ,$\forall j \in \{0, 1, \cdots, k - 1\}, j \neq i, i + 1$. 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 clearly has the additional 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 $r = (n/\frac{\log n}{\log\log n})^{1/\frac{\log n}{\log\log n}} < \log n$. Thus the increase in the routing table size due to the shortcut links is from $r = \log_2 n$ to $r + k - 2 = \log_2 n + \frac{\log_2 n}{\log_2 \log_2 n} - 2$. For example, when there are 2^{20} nodes in the network, this represents an increase from 20 to 23 entries in the routing table.

Note that this topology is not yet suitable for a peer to peer network because it requires a precise number of nodes at each level, functioning correctly at all times for correct routing. The Ulysses network uses a modified version of this topology which can accommodate the dynamics of a peer-to-peer network while preserving the desirable properties of this topology.

IV. Design of The Ulysses Protocol

In this section we present the detailed design of Ulysses that achieves the objectives outlined in Section II. We first present a novel naming and zoning scheme, and a description of the allocation of different portions of the DHT 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, that handles node joins and departures, and enhancements that achieve better topology control without any increase in overheads.

A. *Naming and zoning in the Ulysses network*

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 description that flexibly captures the details of allocation of portions of the hash table to different nodes as well as the dynamics of node joins and departures.

1) Naming convention: In the Ulysses network with k levels and n nodes, each DHT node represents a *zone* in the namespace, 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 $0 \leq l \leq k-1$. The correspondence between the *row identifier* P of a Ulysses node and the k-dimensional row identifier $(x_0, x_1, ..., x_{k-1})$ in a *static butterfly* is as follows : The bits at location $i, i+k, i+2k...$ in P represent x_i in $(x_0, x_1, ..., x_{k-1})$. Thus if $(x_0, x_1, ..., x_{k-1})$ is seen as the location of a node in a k-dimensional cuboid[5] at level l, the bits in P at locations $\{j | j \ mod \ k = i\}$, taken in their order of appearance in P, together form a binary number that corresponds to the location of the node (P, l) in the i^{th} dimension at the level l. In other words, the coordinate of the node (P, l) in the i^{th} dimension, denoted as AP[6]$(P, \{i\})$, is given by the concatenation of every k^{th} bit in the string P starting at the i^{th} location. Formally, in a k level Ulysses network, if $P = a_0a_1...a_l$, $AP(P, \{i\})$, is defined as the string formed by concatenating all a_j such that $j \in$ the arithmetic progression $\{i, i+k, i+2k...\}$. For example if $k = 5$ and $P = a_0a_1...a_{12}$ P, then $AP(P, \{2\}) = a_2a_7a_{12}$.

The expected length of row identifier P of a node in a Ulysses network with n nodes and k levels is $\log_2 \frac{n}{k}$. But the length of P for individual nodes changes due to dynamic arrival and departure of nodes, as explained later in this section.

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, $k \times 2^m >> n$, which is an assumption implicit in all DHT based schemes (e.g. [1], [2], [3], [4], [5]). The AP function can be similarly defined on α as follows: Let $\alpha = a_0a_1...a_{m-1}$ as above. $AP(\alpha, \{i\})$ is the concatenation of all $a_j's$ such that j is an element in the Arithmetic Progression $\{i, i+k, i+2k...\}$.

In a more general form, the function $AP(B, S)$, where B is any binary string and $S = \{i_1, i_2, ..., i_q\}$ is a set of indices, can be defined as the string formed by taking those bits in B that appear at locations j, such that j is in one of the arithmetic progressions $\{i_1, i_1+k, i_1+2k...\}, \{i_2, i_2+k, i_2+2k...\}..., \{i_q, i_q+k, i_q+2k...\}$. The set S is a subset of Ω which is the set of dimensions $\{0, 1, ..., k-1\}$ in a k level Ulysses network. The function $AP(B, S)$ retains the same order of bits as in the string B. For example, if $B = b_0b_1...b_{12}$ and $k = 5$, then $AP(B, \{0, 2, 3\}) = b_0b_2b_3b_5b_7b_8b_{10}b_{12}$.

2) Distributing the Hash Table: The structure of the Ulysses topology is closely related to the way different nodes are allocated different 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 (P, l), 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 $(1011, 3)$. This notion can be extended further by thinking of any binary string P as the set of all binary strings of length m for which P is a prefix. Informally this is equivalent to a partial order on binary strings in which a small string P is a superset of a longer string P' if P is a prefix of P'. 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'. We often represent a node with identifier (P, l) with just its row identifier P when the level l is clear from the context or irrelevant to the discussion. Intuitively, 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 l^{th} level-cuboid. A zone P is said to be a subset of another zone P' if P' is a prefix of P. Note that larger zones have smaller identifiers and vice versa.

[5] A *cuboid* is the generalization of a rectangle in higher dimensions.
[6] AP for its relation to an arithmetic progression.

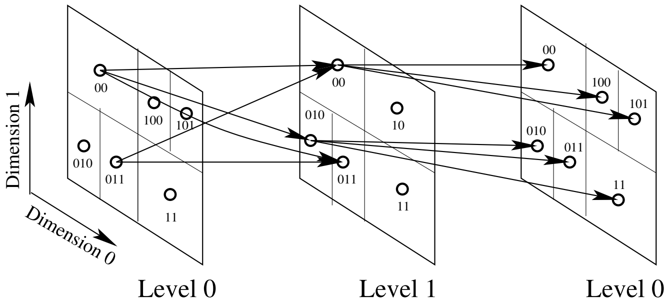


Fig. 2. A Ulysses butterfly with 2 levels and 11 nodes. Unlike Figure 1(b) where all zones of responsibility are homogenous, the arrival and departure dynamics of nodes in the peer-to-peer network cause non-uniformity in the size of the zones of responsibility (areas of regions around the nodes). Links from only 2 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 $\{00,1\}$.

B. *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 III-A. 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 III-B to make the network *edge stress free* and reduce its diameter, are also retained in Ulysses.

Geometrically, the links can be imagined 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 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. These fan-out links correspond to the horizontal and cross links in the butterfly topology.

Formally, 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, $\overline{\{l+1\}} = \Omega - \{l+1\} = \{0, 1, ..., l, l+2, ...k-1\}$.

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 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.

C. 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 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 $l+1$. We say that $(P', l+1)$ "locks" on the destination α along the dimension $l+1$. 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 its 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 an individual 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 (line 5). This can be done locally at the node (P, l) by checking if the bits $AP(P, j)$ are a prefix of the bits in $AP(\alpha, j)$, for each dimension. 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, (i.e., "markovian"). We present a formal proof of the correctness and complexity analysis of routing in section V-A.

D. 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 should be treated as failed temporarily until the repair mechanism kicks in. Otherwise the current node should try to circumvent this failed neighbor by forwarding the query to any of its neighbors while maintaining the condition that the dimensions in S that have been locked already remain locked after this step. Usually there are many such neighbors and the query is forwarded to one of them randomly, thus giving a "random offset" to the query along the next dimension instead of locking it in (line 15). The query will get locked in the remaining dimensions and reach 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".

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 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". After (P, l) gives the query 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, requesting this neighbor to provide a random offset in some dimension j other than $l+1$. This effectively avoids the vicious cycle problem.

1) 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 latency in some cases. A node (P, l) might receive a query for (α, i) such that $AP(\alpha, l+1) \in AP(P, l+1)$ and $AP(\alpha, l+2) \in AP(P, l+2)$. In other words, this key is already locked in the next two levels. In this case, the node (P, l) 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 diameter, it does reduce the average diameter, especially for queries that have to be re-routed due to failures on their paths.

E. Self-stabilization on join of a new node

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

```
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 has locked α in 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 Sec.  IV-D */
15.       else forward the packet to a random neighbor (P',i+1) that locks α on S;
16. end
```

Fig. 3. Algorithm for routing at a node, without the optimization in Sec. IV-D.1.

network. To find such an X, the new node can use any of the discovery mechanisms proposed in the literature, like [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 "$P||0$"[7] and assigns the new node $\mathcal{N}$ the row identifier "$P||1$". Both nodes remain in level l. 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 $P = \mathit{buddy}(P')$ 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, produces a remarkably stable distribution of zone sizes. An ideal mechanism would allow all the nodes to have the same zone-size of $\frac{k2^m}{n}$ where k is the number of levels, n the total number of nodes in the network and 2^m is the key-space at each level.

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 $\mathcal{O}$. However, the nodes in the preceding level $l-1$ that consider $\mathcal{O}$ as a neighbor (referred to as "predecessors"), should be informed of this split[8]. 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 $O(\log n)$.

Another option is not to inform the predecessors through updates at the time of join. A predecessor detects this change when it first tries to route a query through node $\mathcal{O}$ based on its original identifier (P,l), but which should be sent through $\mathcal{N}$, because its identifier $(P||1,l)$ has a better match with the query. The node $\mathcal{O}$ detects this error and returns the query to the predecessor with a routing update. This amortizes the cost of routing updates caused by a join, by distributing the updates across queries, thus reducing the message complexity of a join from $O(\log n)$ to constant.

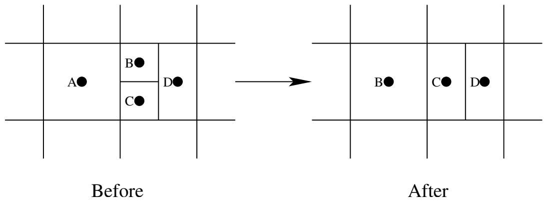


Fig. 4. When buddy of a departing node (A) is split into multiple zones: the node with smallest zone (B) is *promoted* to take over departing node's zone and C merges with B's zone.

F. 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*, cleaning up 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.

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. These zones can 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 original buddy may be 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 small nodes will be "promoted" to take over the departing node's zone and the other node will merge with its original buddy. 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

[7] $||$ refers to the concatenation operation.

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

query for a key, issued by the departing node, and held by one of the smaller nodes in the buddy zone.

Like in 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 section IV-G. The newly "promoted" node should do the same thing about its own departure from its original zone, thus increasing the total expected complexity of a departure by a factor of 2.

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/housekeeping phase. The detection itself might occur either by a *time-out* 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. Time-out 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 a Ulysses network.

Once a node has detected the ungraceful departure of one of its neighbors, it initiates and carries through all the housekeeping operations on the 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.

G. 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 the following. In both Chord and Ulysses, each join/departure affects about $\log n$ other nodes that have a link to the nodes affected by the join/leave, and they all need to be notified. In Ulysses, because the "coordinates" of these neighbors are very close to the affected nodes, a property of 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].

H. Heuristics to make the zones of responsibility uniform

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 section IV-E.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. While one approach checks for existence of a larger zone among the neighbors of the destination node, the other approach examines all zones along the path traversed by the join request. Neither approach requires any extra cost of storage or communication than the default join procedure. Our simulation experiments with dynamic arrivals and departures indicate that over 90% of the nodes end up with routing tables of size around $\log_2 n$, while the routing table sizes of rest of the nodes are within the interval $[\frac{1}{2}\log_2 n, 2\log_2 n]$ [14].

I. 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 comparisons with other DHT-based protocols [1], [2], [3], [4], we configure k such that the routing table size of Ulysses is no more than $\log_2 n$. Given a network size n, we define $\mathcal{K}(n)$ as the smallest k value that results in a routing table size no more than $\log_2 n$, i.e., $n \leq k(\log_2 n)^k$. In other words, given the routing table size of no more than $\log_2 n$, setting k to $\mathcal{K}(n)$ allows Ulysses to achieve minimum diameter. The following formula shows the values of $\mathcal{K}(n)$ on various ranges of n.

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

Although ideally the number of levels should be $\mathcal{K}(n)$ when the network size is n, we do not expect Ulysses to have to adapt k to the changing values of n dynamically. Instead, a value for k would be chosen a priori based on off-line information regarding the approximate size ranges of the peer-to-peer network. Such a fixed k will not cause problem for Ulysses for two reasons. First, the above formula shows that a given value of k can be optimal for a wide range of values of n (e.g., from $n = 256K$ to $n = 16M$ when $k = 5$). 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 and decrease slightly. In other words, the performance of a Ulysses network with n nodes is quite insensitive to the choice of number of levels and even non-optimal choices do not impact performance significantly[9]. In all our experiments reported in Section VI, we select the number of levels of Ulysses network to be $\mathcal{K}(n)$.

V. 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 state the key topology invariants that Ulysses maintains, and 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 of such "repair" is $O(\log n)$ as compared to $O(\log^2 n)$ in Chord. 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) $H(P, i, j) = \{R(\alpha, j) | \alpha \in P\}, 0 \leq j \leq k-1, j \neq i, i+1$. Here we simply formalize our intuitive characterization of Ulysses' connectivity in Section IV-B and Fig. 2.
3) *Progress statement.* Assume that both the zoning and the connectivity invariants hold. Then our routing algorithm (shown in Fig. 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 Section V-C.
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 Fig. 3, with high probability. This property is formally stated in Theorem 3.

[9]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 $n^{\log\log n / \log n} = \log n$.

We introduce some definitions and notations that will be used in later expositions. Let a key α be $a_0 a_1 ... a_{m-1}$. Recall that $AP(\alpha, \{j\})$ is the concatenation of all a_i's such that i is an element in the Arithmetic Progression $\{j, j+k, j+2k, ...\}$. $AP(\alpha, S)$ where $S \subseteq \Omega$, and $AP(P, S)$ where P is a zone, are defined similarly before. 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 $merge(\alpha, S, \beta)$ as a "hybrid" of two keys $\alpha = a_0 a_1 \cdots a_{m-1}$ and $\beta = b_0 b_1 \cdots b_{m-1}$ 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 $a_0 b_1 a_2 a_3 b_4 a_5 b_6 a_7 a_8 b_9 a_{10} b_{11} a_{12}$.

A. Progress and Termination

We first introduce a 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 $S_1 \bigcup S_2$.

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 $\gamma = merge(\beta_1, S_1, \beta_2)$. 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 ($(i+1)^{th}$) 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 Fig. 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 $\gamma = merge(\alpha, S \bigcup \{i+1\}, \beta)$. 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, $l \leq k$, at least l dimensions have been locked.

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

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 Fig. 3), the query will follow a shortcut link to the destination, the existence of which is due to the connectivity invariant. ■

B. Robustness

Our randomized detour algorithm minimizes the penalty of detour. After the "detour", at most 0, or when "vicious cycle avoiding" is needed 1, 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 three to three additional hops. The following theorem captures this intuition. We omit its proof since it is quite straightforward from our description in Section IV-D.

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 IV-D, it will land on a P'' such that $L(\alpha, P'') = \Omega - \{i+1, i+2\}$, thereby avoiding the vicious cycle as discussed in IV-D (2) Otherwise, after on one randomized step, it will land on a node P'' such that $L(\alpha, P'') = L(\alpha, P)$.

C. 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 Sec. IV-E and IV-F, 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 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, $O(\log n)$ 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.

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 $\{R(\alpha, i-1) | AP(\alpha, \overline{\{i\}}) \in AP(Q, \overline{\{i\}})\}$, which we denote as $Pred(Q, i)$. The expected size of $Pred(Q, i)$ 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, $\forall (Q', i-1) \in Pred(Q, i)$, either $AP(Q', \Omega - \{i\})$ is a substring of $AP(Q, i)$ or vice versa. Since the members of the set $Pred(Q, i)$ 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 Section IV-D.1, 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.

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.

VI. 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 Section VI-A, 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 Section VI-B, 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 Section VI-C, we show that Ulysses is very robust. Even when a large percentage 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[10] 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 million. Our implementation of the Chord protocol is from the description given in [1]. The following sections describe our experiments and analysis of the results.

A. Query path length

Our simulation results verify our expectation from the theoretical analysis that Ulysses achieves significantly lower 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 IV-I, 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 Fig. 5. 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 IV-I. As expected, curves for Chord coincide with functions $\log_2 n$ for the worst-case and $\frac{1}{2}\log_2 n$ for the average case. They look "linear" in the figure since the x-axis is in log scale. We can see from Fig. 5 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[11] of Ulysses compared to Chord. Figure 6 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.

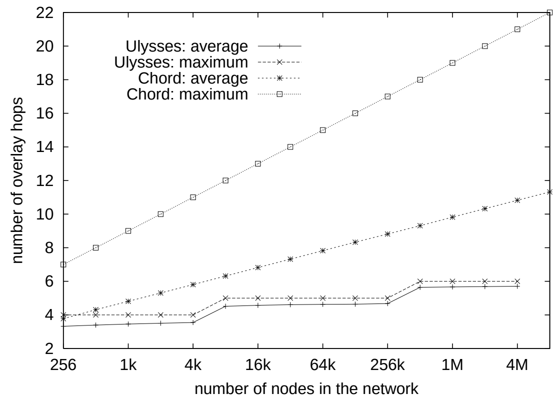


Fig. 5. The average and maximum number of hops required for a query to be routed correctly to its destination in an overlay network with n nodes.

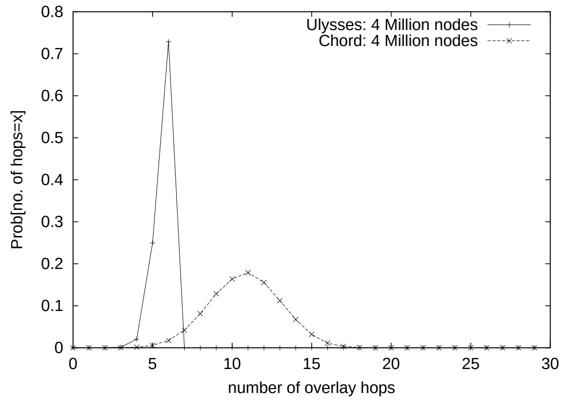


Fig. 6. The probability density function of number of hops required for a query to be routed correctly to its destination in an overlay network with 2^{22} nodes.

B. 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

[10] Proposed enhancements to protocols like CAN [5], Pastry [3] 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.

[11] 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 [14].

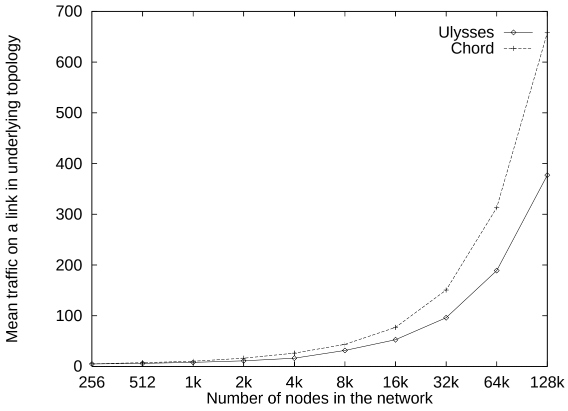


Fig. 7. Variation of traffic density with different sizes of overlay networks.

(i.e., "stays in the network") for less number of overlay hops. In this section, 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 [15]. 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 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 7 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.

C. 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 section VI-A is simulated here. The probabilities reported in these plots on robustness are computed from the results of simulating 10^5 queries in each case.

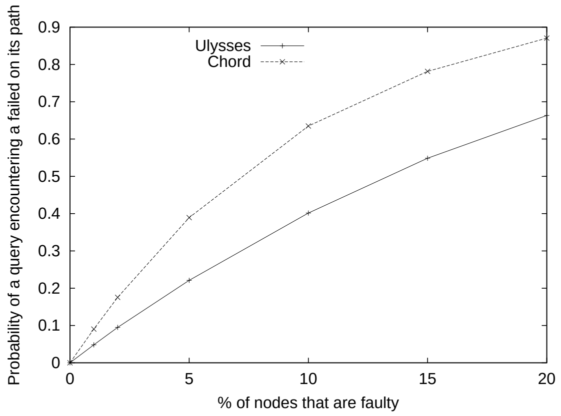


Fig. 8. Probability that a query encounters a node failure as we vary the percentage of node failures in a 4 million node overlay network.

Figure 8 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.

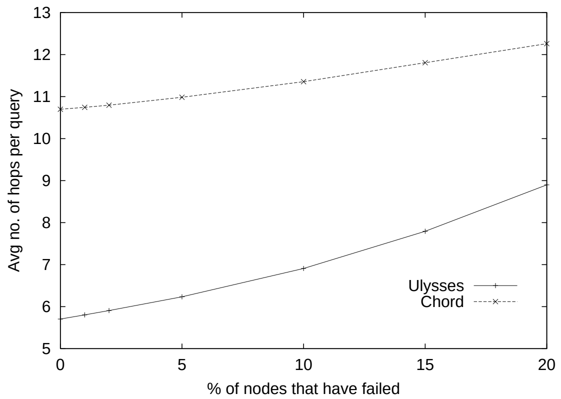


Fig. 9. Average number of hops required to route a query vs. percentage of failed nodes in a 4 million node network.

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 IV-D. Figure 9 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. However, 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 path. The larger slope for Ulysses is due to the aforementioned higher penalty for routing around failures in a Ulysses network. 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.

VII. 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 might cause up to $O(\log_2^2 n)$ nodes between two butterfly nodes on the last level, thus taking the worst case diameter to $O(\log_2^2 n)$. For similar reasons, the maximum *congestion* 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 [16], 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 known whether Koorde can *self-stabilize* like Chord or Ulysses. 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 [17] propose a similar mechanism to use the de Bruijn graph to achieve a diameter of $O(\log n/\log\log n)$ 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 et al. require the number of neighbors to be $\log n$. This requires an *a priori* knowledge of the logarithm of network size. Ulysses on the other hand requires a knowledge of "$\log n/\log\log n$" which is much less sensitive to changes in network size.

VIII. 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 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, a buddy-based protocol for assigning responsibility for portions of the key space and handling self-stabilization, and a collection of robustness techniques to allow efficient and correct operation under network dynamics.

Our future work will focus on two directions. First, Ulysses is particularly well suited for efficient multicast using wildcards in the node addresses. Such a multicast operation has importance in the housekeeping operations associated with maintaining the overlay, and may also be useful as an overlay service. Second, we plan to implement and test Ulysses in wide-area emulation and testbed environments.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of ACM SIGCOMM '01*, 2001.

[2] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep., U.C. Berkeley Tech. Report UCB/CSD-01-1141, 2001.

[3] A. Rowstron and P. Druschel, "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] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," in *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM '01*, 2001.

[6] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," in *Proc. of ACM SIGCOMM '02*, 2002.

[7] "IRIS: Infrastructure for Resilient Internet Systems," http://www.project-iris.net.

[8] S. Ratnasamy, S. Shenker, and I. Stoica, "Routing Algorithms for DHTs: Some Open Questions," in *Proc. of 1st Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[9] J. Xu, "On the Fundamental Tradeoffs between Routing Table Size and Network Diameter in Peer-to-Peer Networks," in *to appear in Proc. of IEEE Infocom 2003*, 2003.

[10] L. Kleinrock, *Queueing Systems*, vol. I and II, J. Wiley and Sons, 1975.

[11] H. J. Siegel, "Interconnection networks for SIMD machines," *Computer*, vol. 12, no. 6, 1979.

[12] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," in *Proc. of ACM PODC*, 2002.

[13] P. Francis, "Yoid: Extending the Internet Multicast Architecture," Unrefereed report, 38 pages, Apr 2000.

[14] A. Kumar, S. Merugu, J. Xu, E. Zegura, and X. Yu, "Ulysses: A Robust, Low-Diameter, Low-Latency Peer-to-Peer Network," Tech. Rep. GIT-CC-03-30, College of Computing, Georgia Institute of Technology, 2003.

[15] E. Zegura, K. Calvert, and M. J. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, Dec 1997.

[16] M. F. Kaashoek and D. R. Krager, "Koorde: A simple degree-optimal distributed hash table," in *IPTPS*, Feb 2003.

[17] M. Naor and U. Wieder, "A Simple Fault Tolerant Distributed Hash Table," in *IPTPS*, Feb 2003.