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,
{akumar, merugu, jx}@cc.gatech.edu

Xingxing Yu
School of Mathematics,
Georgia Institute of Technology,
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. In this paper, we present Ulysses, a peer-to-peer network based on the butterfly topology that achieves 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* $\log n$*. 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., [15, 18, 12, 9, 10]). These protocols generally rely upon 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.,[14, 1]).

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. 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, savings in query hops are magnified when one considers the network layer impact.

It was observed in [11] 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 [15, 18, 12, 9]), or (2) a routing table of size d and network diameter of $O(dn^{1/d})$ (including [10]). It was asked in [11] 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 [16] 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 [16] 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 most existing schemes[1]. This

* This paper was supported in part by the National Science Foundation under Grants ANI-0113911, ANI-0081557, DMS-0245530 and under NSF CAREER Award Grant ANI-0238315, and by the NSA under Grant MDA-904-03-1-0052.

1 In section 6, we discuss some recently proposed schemes that

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] [15] 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 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 [15]. By Little's Law [5], 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 [13]. 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 challenges. 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, 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 solution approach in Section 2. After introducing the static butterfly topology and its shortcomings in Section 3, we describe Ulysses in detail in Section 4. In Section 5, we discuss results of our performance evaluation by simulation. Section 6 has other related work followed by concluding remarks in Section 7.

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.

achieve the bound of $O(\frac{\log_2 n}{\log_2 \log_2 n})$.

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 $O(\log n)$.

2.1. Design Objectives

Low latency routing: Ulysses tries 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 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 about $\log n$ at each node. Both properties need to be maintained despite node joins and departures.

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.

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 DHT based peer to peer networks have only used the constant out-degree version of this topology that achieves a diameter of $O(\log n)$ [7]. 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 4 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 4.4 and 4.5. A salient feature of Ulysses' self-stabilization mechanisms is their low cost, discussed further in section 4.5.3.

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. Our simulation results show 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.

3. The static butterfly topology

The structure of Ulysses is inspired from the well known butterfly topology [13]. 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 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 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 $(x_0, x_1, \cdots, x_{k-1}; i)$, 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 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.

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

In [16], the notion of *edge congestion free* is defined under the assumption of *uniform traffic load*[4]. To avoid confusion with 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. Such edges are called *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 more traffic than a cross link.

3.2. Shortcut links to remove stress

We solve this stress problem by adding $k-2$ "shortcut" links from the node $(x_0, x_1, \cdots, x_{k-1}; i)$ at level i to nodes $(x_0, x_1, \cdots, x_{k-1}; j)$ at level j, $\forall j \in \{0, 1, \cdots, k-1\}, j \neq i, i+1$. Thus, 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.

This topology is not yet suitable for a peer to peer network because it requires a precise number of nodes

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. In addition, if the sources of queries are themselves assumed to be distributed uniformly, the resulting scenario is one of *uniform traffic load.*

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.

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 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. The section ends with a description of the self-stabilization mechanism of Ulysses, which handles node joins and departures.

4.1. 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.

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 $0 \leq l \leq k-1$. 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 $(x_0, x_1, ..., x_{k-1})$. In other words, the coordinate of the node (P, l) in the i^{th} dimension, is given by the concatenation of every k^{th} bit in the string P starting at the i^{th} location. For example if $k = 5$ and $P = a_0 a_1 ... a_{12}$, then $a_2 a_7 a_{12}$ is the coordinate of P in 2^{nd} 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 $\log_2 \frac{n}{k}$. 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. [15, 18, 12, 9, 10]).

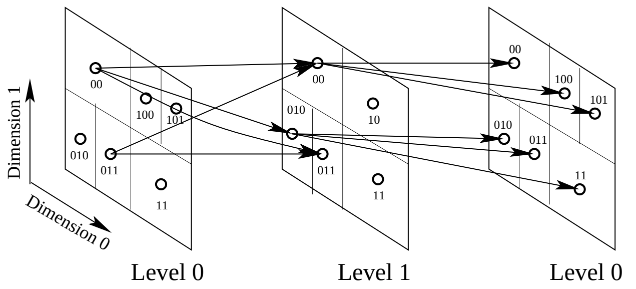

Figure 1: **A Ulysses butterfly with 2 levels and 11 nodes. Links from only 2 nodes in each level are shown for clarity. This figure is "wrapped around" with level 0 shown twice.**

4.1.2. Distributing the Hash Table The structure of Ulysses 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, 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'.

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.

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 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 3.2 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 1, 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. Returning to Fig. 1, 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) has links to all nodes (P', j) such that either P is a prefix of P' or vice versa, for any 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.

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 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 complete. 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 2. 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 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 complete (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").

4.3.1. Robustness Due to dynamic departures, a node (P, l) might identify $(P', l+1)$ as the next hop for a query, just after $(P', l+1)$ has departed but before the self-stabilization mechanism of Ulysses has repaired this fault. In such a situation the current node (P, l) needs to "route around" this failed neighbor. This is achieved in Ulysses by simply requiring the node (P, l) to forward the query to any of its neighbors in level $l + 1$ while maintaining the condition that the dimensions in S that have been locked already remain locked after this step. This simple "detour" mechanism is adequate to correctly reroute most queries that encounter faulty nodes on their paths. A more complete description of the robustness features of Ulysses, including the detour mechanism and a vicious cycle avoidance mechanism that together guarantee successful rerouting of all queries destined to alive nodes, is presented in [6].

4.3.2. 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 latency in some cases. 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 diameter, it does reduce the average diameter, especially for queries that have to be re-routed due to failures on their paths.

4.4. Self-stabilization on join of a new node

4.4.1. 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 [3]. 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

```
1.  Routing algorithm at a node $(P, i)$
2.  input: a query key $(\alpha, j)$
3.  output: a ‘‘next-hop’’ decision
4.  begin
5.     Compute the set of dimensions $S$ on which $P$ has locked $\alpha$
6.     if ($S$ == $\Omega$) 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 $\alpha$ in on the set $S \bigcup \{i+1\}$;
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 (($S == \Omega - \{i+1\}$) && ($i+1 == j$)) then nothing can be done; /* the destination is faulty */
13.       else forward the packet to a random neighbor $(P', i+1)$ that locks $\alpha$ on $S$;
14. end
```

Figure 2: **Algorithm for routing at a node, without the optimization in Sec. 4.3.2.**

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 the concatenation operation. 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 even distribution of zone sizes. Note that in an ideal situation, all the nodes would 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.

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

4.5. Self-stabilization on Node Departures

Nodes can depart from a Ulysses network at will, at any point in time. Ideally, a departing node should leave *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.

4.5.1. 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, these two nodes must have zones of equal size. These zones can be merged to create a zone twice the size. In case the buddy's zone has split into multiple smaller zones, a more sophisticated protocol, discussed in [6], can be used to identify a node with one of these smaller zones. This node will then be "promoted" to take over the zone of the departing node.

Similar to the case of a 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 later in section 4.5.3.

4.5.2. 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 finds out 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 a more suitable candidate for 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 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.

4.5.3. 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 [5].

4.6. 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, 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 value of k, for which the routing table size does not exceed $\log_2 n$, i.e., $n \leq k(\log_2 n)^k$. 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:

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

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[5]. For our experiments reported in the next section, we select the number of levels of Ulysses network to be $\mathcal{K}(n)$.

5. Performance Evaluation

We have evaluated various aspects of the performance of Ulysses using simulation. The findings of our simulation study can be summarized as follows: In Section 5.1, we show the reduction in worst-case and average path lengths of routes taken by queries, achieved by Ulysses. In Section 5.2, 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. Section 5.3 discusses simulation results demonstrating the robustness of Ulysses.

We choose Chord [15] 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[12] and Tapestry[18] 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 re-

5 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 $n^{\log \log n / \log n} = \log n$.

sults can be extended to larger bases as well. Similarly, we do not consider the class of protocols (e.g. CAN [10]) 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[6] of Ulysses is to achieve a low diameter, such protocols would compare 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 as the routing of 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 [15]. The following sections describe our experiments and analysis of the results.

5.1. 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 4.6, 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. 3. Two curves are plotted for each protocol, denoting respectively the worst-case and the average query path lengths measured in the number of overlay hops. The "steps" on both Ulysses curves are due to the increase of parameter k when n becomes larger, as discussed in Section 4.6. 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. 3 that both the average and maximum number of hops required by Ulysses are less than the

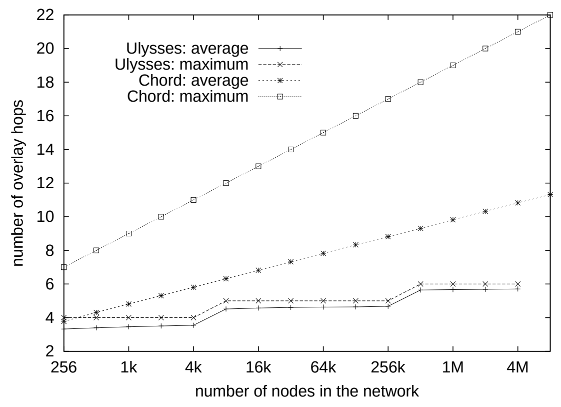

Figure 3: **The average and maximum number of hops required for a query to be routed correctly to its destination in an overlay network with n nodes.**

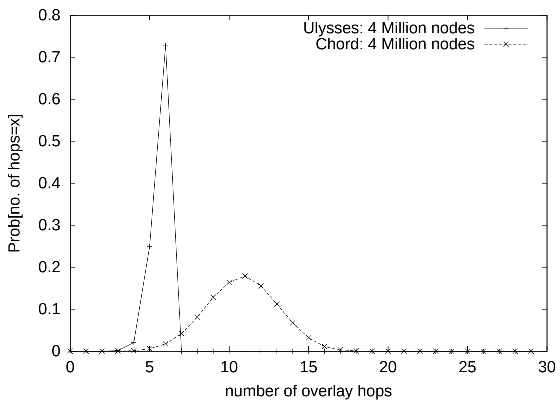

Figure 4: **The probability density function of number of hops required for a query to be routed correctly to its destination in an overlay network with 2^{22} nodes.**

average number of hops required by Chord. The difference is more pronounced at a larger network size. These differences highlight the better scaling properties[7] of Ulysses compared to Chord. Figure 4 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

6 Proposed enhancements to protocols like CAN [10], Pastry [2] 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.

7 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 [6].

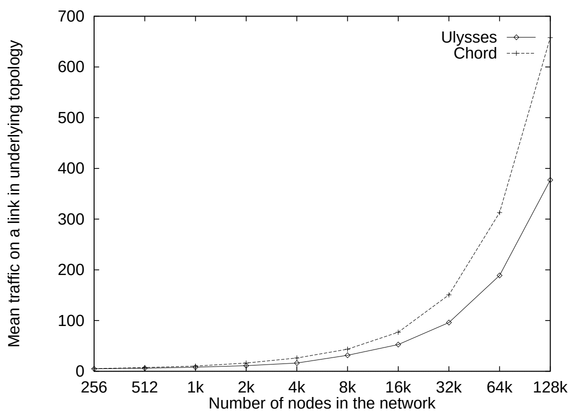

Figure 5: **Variation of traffic density with different sizes of overlay networks.**

its low variance, indicate that the length of most of its paths is very close to the mean.

5.2. 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. As a consequence, the amount of traffic that goes through each link and node also goes down proportionally. In this section, we demonstrate this advantage through simulations.

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 [17]. 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 5 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.

5.3. Robustness

The robustness of Ulysses is evaluated in comparison with Chord by simulating sudden ungraceful departures of a fraction of the node population.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 5.1 is simulated. The results, not shown here due to lack of space, demonstrate that almost all queries destined for nodes that are "alive", are routed correctly. Furthermore, our simulations show that queries being routed in a Ulysses network have a significantly smaller probability of encountering a failed node on their path. We also observe that Ulysses retains its reduction in average path length of queries even in the face of massive node failures. Figures demonstrating these results, along with a detailed discussion, are available in [6].

6. Other Related Work

The Viceroy network [7], 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 [15] 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 *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 [4], 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 [8] 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 significantly outnumber the 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 (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.

7. 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.

References

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

[2] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Distributed Hash Tables. In *International Workshop on Future Directions in Distributed Computing*, 2002.

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

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

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

[6] A. Kumar, S. Merugu, J. Xu, E. Zegura, and X. Yu. 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.

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

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

[9] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proc. of ACM SPAA*, 1997.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, 2001.

[11] 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.

[12] 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.

[13] H. J. Siegel. Interconnection networks for SIMD machines. *Computer*, 12(6), 1979.

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

[15] 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.

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

[17] E. Zegura, K. Calvert, and M. J. Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6), Dec 1997.

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