DONUT: Building Shortcuts in Large-Scale Decentralized Systems with Heterogeneous Peer Distributions

Sergey Legtchenko, Sébastien Monnet and Pierre Sens
LIP6/UPMC/CNRS/INRIA
4 Place Jussieu - 75005
Paris, France
Email: Firstname.Name@lip6.fr

***Abstract*—Large-scale distributed systems gather thousands of peers spread all over the world. Such systems need to offer good routing performances regardless of their size and despite high churn rates. To achieve that requirement, the system must add appropriate shortcuts to its logical graph (overlay). However, to choose efficient shortcuts, peers need to obtain information about the overlay topology. In case of heterogeneous peer distributions, retrieving such information is not straightforward. Moreover, due to churn, the topology rapidly evolves, making gathered information obsolete. State-of-the-art systems either avoid the problem by enforcing peers to adopt a uniform distribution or only partially fulfill these requirements.**

To cope with this problem, we propose DONUT, a mechanism to build a local map that approximates the peer distribution, allowing the peer to accurately estimate graph distance to other peers with a local algorithm. The evaluation performed with real latency and churn traces shows that our map increases the routing process efficiency by at least 20% compared to the state-of-the-art techniques. It points out that each map is lightweight and can be efficiently propagated through the network by consuming less than 10 bps on each peer.

***Keywords*-churn; overlays; long-range links; Small-World graphs; routing; range queries; heterogeneous keyspaces;**

I. Introduction

Over the last few years, several widespread large-scale applications adopted a highly decentralized architecture [1], [2], [3]. In such systems, the load is spread among the hosts, which constantly need to interact using an overlay in order to retrieve shared *resources* (information or services). While looking-up for a resource in the overlay, peer's request has to be forwarded (*i.e.,* routed) to several peers before reaching its destination peer. As most of the distributed applications have latency constraints, the resource discovery process must be efficient. This implies minimizing the average graph distance in the overlay[1].

To do that, shortcuts or *long-range links* are added to the base-graph of the overlay. Kleinberg has shown that the optimal routing process is achieved in a grid if the probability for a peer p to choose a peer q as its long-range link depends on the graph distance between p and q. Kleinberg's distribution of the shortcuts is called *d-harmonic* [4]. Therefore, to reach this distribution, a peer needs to find peers that are at the appropriate graph distance in the overlay. It implies for the peer to have some global information about the overlay graph.

However, in large-scale distributed systems, having an accurate global view of the topology is generally impossible, because the number of participating entities is too large. Furthermore, the graph constantly changes due to churn (*i.e.,* peer connections and disconnections). Therefore, peers have to enable a mechanism that approximates the graph distance without locally maintaining the exact continuously changing topology.

This problem has been mainly addressed by the Distributed Hash Tables (DHTs [5], [6], [7]). Some DHTs manage to build long-range links enabling efficient routing. However, these systems rely on the use of a hash function to assign peer/resource identifiers, which makes the identifier distribution uniform but breaks semantic relationships between resources. Yet, such relationships are necessary for native range query support, a widespread application requirement [8], [9].

A few works, *e.g.,* Mercury [10] and more recently Oscar [11], addressed the issue without the use of a hash function. However, the proposed solutions present several drawbacks. The main concern is that they do not monitor the global system state. Yet, the system topology is constantly changing under the churn impact. Therefore, these solutions can only adapt afterwards by re-probing the system. Moreover, their probing mechanism is based on random walks, and is thus suitable only for expander graphs [10]. Oscar, the most efficient of these systems is detailed in the next section and is then compared to our contribution (see Section V).

In this paper, we propose DONUT[2], a mechanism that provides, on each peer, a global map which allows to locally estimate the graph distance to other peers. The main idea is to either use existing lookup messages to piggyback information or to provide a simple gossip algorithm in order to construct, on each peer, a fuzzy view of the whole system peer distribution. This map is then used to build efficient long-range links. Thanks to DONUT, the long-range rewiring process is very lightweight, because the localization of appropriate peers is made *locally* by using the map. In case of modifications due to churn, the local map progressively adapts itself to reflect the new density distribution, allowing peers to replace

[1]In the rest of the paper, we call *graph distance* the minimal distance, in number of hops, between two peers in the overlay.

[2]DONUT: Density-aware Overlay for Non-Uniform Topologies.

obsolete long-range links by new ones before performance degrades.

To the best of our knowledge, there exist no algorithm to build long-range links based on a map that approximates the global distribution of peers in the system. Yet, our map may also be useful to other distributed system mechanisms, such as efficient global load balancing, system monitoring, network size estimation, etc. Therefore, the contributions of this work are: 1) a distributed algorithm that locally provides a fuzzy map of the whole system distribution and 2) an algorithm that uses the map to build efficient long-range links.

A detailed evaluation of DONUT was performed in a discrete event simulator with the use of traces collected from existing distributed systems to simulate realistic latencies and churn (see Section V). The lessons learned from the work are:

1) DONUT accurately estimates the graph distances between peers, providing an optimized routing process in heterogeneous peer distributions.
2) The maps are lightweight: average map size is of 2.2 Kbytes for 2500 peers in our evaluation.
3) Efficient propagation of the map between peers may be achieved with less than 10 bps of traffic on each peer.

The rest of this paper is organized as follows: Section II describes the needed background and discusses the related work. DONUT, our contribution is described in Sections III and IV: the former describes the density map construction and the latter details the shortcut-wiring process. Section V presents the evaluation before Section VI concludes.

II. Background and Related Work

Decentralized resource discovery is one of the major issues of large scale distributed systems. In order to offer a satisfying performance, the discovery process should exhibit two fundamental properties: 1) If the overlay contains a resource, the process must necessarily find it[3]; 2) the process must be efficient: the completion time and the number of exchanged messages must be at worst polylogarithmic. Early attempts to address the problem failed to satisfy these requirements. In Gnutella, the resources were discovered by flooding the overlay with search messages [12]. This approach is costly in terms of messages and may fail since the flooded messages have a limited time to live.

The first step to an efficient decentralized discovery mechanism was the Key Based Routing layer (KBR [2], [5]). Resources and requests are assigned a *key* in an d-dimensional keyspace and each peer is assigned an identifier in the same keyspace. In the rest of the paper, we define keys of resources and identifiers of peers as their *coordinates* in the keyspace. The distance between any couple of keyspace-coordinates should be computable, *i.e.,* the keyspace must be a metric space. We note *keyspace distance* the euclidean distance between two coordinates of the keyspace. The peer which identifier is the closest to a resource key is called the *root* of the resource and is responsible for its storage.

However, one must find a way to locate that peer inside the overlay. In other words, the overlay must ensure that a decentralized greedy algorithm is always able to route a message to the resource, by knowing only its coordinates in the keyspace. The greedy routing strategy always chooses as the next hop the peer that minimizes the distance to the target coordinates [13]. The routing algorithm stops on the root of the target coordinates. To simplify the routing process and the overlay structure, most of the keyspaces are *borderless*, which means that for each dimension of the keyspace, the minimal value is close to the maximal value, and the routing is performed with values reduced *modulo* the size of each keyspace dimension.

Several topologies are able to support a keyspace, depending on its dimension. For example, a ring enables decentralized greedy routing for one dimensional keyspaces. Topologies based on Delaunay triangulations or Voronoi tessellations ([14], [15]) provide greedy routing for keyspaces of two or more dimensions [13]. Some more randomized topologies also exhibit the same property w.h.p. ([16], [17]). We call *KBR-overlays* decentralized systems which topology is able to support a keyspace. A KBR implemented on an appropriate topology ensures the first requirement of resource discovery.

The couple formed by a keyspace and its KBR-overlay exhibits several important properties. 1) Each peer p is *responsible* for a well delimited region of the keyspace formed by the coordinates that are closer to the coordinates of p than to other peers. 2) The *graph distance* between two coordinates of the keyspace is equal to the distance in hops between the two peers responsible for the coordinates. 3) By definition all coordinates have a root. Therefore, the responsibility zones of the overlay peers form a partition of the keyspace. 4) There is a correlation between the keyspace distance and the graph distance: during the greedy routing process, each hop decreases the keyspace distance *and* the graph distance to the target.

The second requirement of efficient resource discovery is known to be satisfied only for topologies that have small characteristic path lengths, *i.e.,* that have natural shortcuts in their graph [18]. Such topologies are called "Small-world" graphs by analogy with the small-world phenomenon [19].

Unfortunately, the topologies listed above do not belong to that class of graphs. To enable the small-world property, shortcuts need to be added to the original topology. Therefore, each peer maintains a set of long-range links. Kleinberg showed that, to obtain optimal routing properties in a d-dimensional keyspace, the long-range links have to follow a d-harmonic distribution. That is, for a bi-dimensional space, let p be the peer that is choosing the links, and $GD(i, j)$ the *graph distance* between peers i and j. If the links l of p are chosen with a probability proportional to $GD(l, p)^{-2}$, the link distribution is optimal and the routing process is polylogarithmic [4].

[3]Its contrapositive is also important: a negative result must signify that the resource does not exist in the overlay.

In practice, the d-harmonic distribution may be difficult to build. Indeed, unlike keyspace distances, graph distances between peers are not straightforward to obtain locally without having a real-time copy of the whole topology. Fortunately, there is a correlation between the keyspace distance and the graph distance between peers. In fact, in case of uniform distribution, the keyspace distance is proportional to graph distance on average.

DHTs such as PAST [6] or DHash [7] take advantage of that proportionality to build long-range links that enable logarithmic routing. These systems are built on a ring-based KBR-overlay which assumes such a uniform distribution of keys (ensured in practice by the use of hash functions to generate keys for both data blocks and peers).

However, as keys are distributed according to some hash function, the resources that are semantically close may be far from each other in the keyspace. In other words, the semantic relationship between resources is lost, which complicates the retrieval of resources located in some semantic *range*. Such multiple-resource requests are called *range queries* and are necessary to many distributed applications[4]. To natively support range-querying, the keyspace needs to be *semantic*. In that case, the resource coordinates are not chosen by a hash function, but defined by some semantic properties of the resource. In the rest of the paper, we call *semantic distance* the keyspace distance.

Unfortunately, many studies show that distributions of existing semantic keyspaces are highly heterogeneous. For instance, research on Massively Distributed Games shows that the keyspace contains popularity hotspots where almost all peers are gathered, whereas large portions of the keyspace are almost deserted. The extreme non-uniformity of the keyspace is also confirmed by studies about file sharing peer-to-peer systems such as Gnutella [20]: exchanged file keys tend to follow a Zipf distribution ([11], [21]).

In these highly non uniform distributions, the semantic distance is no longer proportional to the graph distance. Yet, it is still possible to exploit semantic distance to build a d-harmonic distribution. For that, the peer must be somehow aware of the keyspace distribution. It has been proven that the Kleinberg approach can be used in non uniform keyspaces if the distribution function of peers in the keyspace is known [22].

Several systems are implicitly incorporating knowledge about the density distribution in the long-range rewiring process. Mercury [10] and Oscar [11] probe the keyspace, making an approximation of its density. These works are the closest to ours. We now focus on Oscar because it proposes a more accurate probing mechanism than Mercury [11].

Oscar [11] uses a local algorithm (that we call the *log-partition algorithm*) to form $log_2(n)$ sets of the peer population (see Figure 1). Let p be the peer executing the algorithm. All peers are locally sorted according to their graph distance to p. Peers which distance to p is bigger than the median form the first set of the partition (the set 1 in Figure 1). Then, the set of peers which distance to p is lower than the median is halved the same way. The process is repeated until the subset contains only the close range neighbors of p (the set 4 of Figure 1 is the last step of the algorithm). Finally, p chooses uniformly at random a long-range link in each subset. The formed set of links follows the Kleinberg's distribution [11].

Since it is impossible for p to retrieve information about the graph distance to all peers, Oscar estimates the subset's population by using random walks. At each step of the partition algorithm, it performs a constant number c of bounded random walks inside the current interval. The result after $c \times log(n)$ bounded random walks is a partition of the keyspace built by taking into account the distribution of peers. However, no global peer distribution is actually built. Evaluations of Oscar described in [11] show that

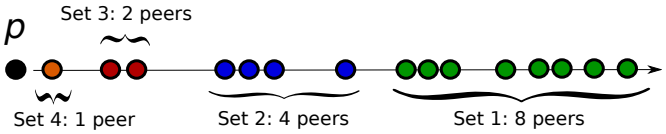

Figure 1. Population partition formed by the log-partition algorithm.

it outperforms other systems over heterogeneous-keyspace overlays. Yet, in case of churn, Oscar is not able to detect the modifications of the density distribution until a lookup performance degradation is perceived. This happens because no keyspace monitoring is performed. Furthermore, Oscar's approach (as well as Mercury) is limited to expander graphs [10], because it heavily relies on random walks to perform the population estimations. It is known that random walks over a graph are guaranteed to be rapidly mixing (*i.e.,* to converge to a uniform distribution in a polylogarithmic number of hops) only for expander graphs [10].

III. Building a global keyspace density map

This section describes the algorithms used to build a keyspace density map.

Overview. On each DONUT-peer, the map is implemented by using a tree that locally indexes each region of the map. Since we choose to implement a bi-dimensional keyspace to illustrate our contribution, we need to use a quadtree to correctly index all the regions of the map[5]. Each node of the quadtree is responsible for a square region of the map. A leaf of the tree may engender four children representing the four cardinal directions. In that case, its region is split in four equal subsquares, and the responsibility for each subsquare is given to the corresponding child (*e.g.,* the upper left square is given to the North Western child). Thus, the union of the subsquares forms a partition of the keyspace (see Figure 2). Each leaf of the tree also contains information about the peer density inside the region it is responsible for. When a leaf

[4] *E.g.,* a player of a multiplayer game may need to retrieve all the objects located inside a specified area of the game-map [9].

[5] For instance, one dimensional spaces only require a binary tree, while 3D spaces need an octree.

is split, its children inherit its density value. Therefore, the set of the quadtree's leaves forms an approximation of the keyspace density distribution function.

At the bootstrap, a peer holds no information about the keyspace distribution: the map has to be filled with density information. There are two possible ways for a peer p to complete the map: 1) use local information and 2) receive information from other peers.

At the beginning, the only available information is local information. As the peer joins the overlay, it is assigned coordinates in the keyspace by the distributed system and is routed to its location in the overlay according to these coordinates. During the process, it obtains the coordinates of its overlay neighbors. Thanks to that, it is able to determine the density of the keyspace that contains its coordinates and its neighbors' ones. The density is obtained by computing the surface of a circular portion of the keyspace centered on the peer's coordinates $coord$, with a radius rad equal to its distance to the farthest neighbor's location. This area is then divided by the number of neighbors of the peer to obtain a density d. The triplet ($coord$,rad,d) forms the local information about the keyspace density.

After the insertion of the local information, every peer locally stores a map of the keyspace with an estimation of its surrounding density (see Figure 2.left). The union of all the peer maps forms an accurate density mapping of the whole keyspace (see Figure 2.right). Therefore, to fill their maps, peers need to exchange their local information.

The first step of the map completion is performed by Algorithm 1, and the insertion of the information received from other peers is done by Algorithm 2. It is important to highlight that once the density information has been received, no operation is performed on the network. All the steps of Algorithms 1 and 2 are executed *locally* and *nodes* composing each quadtree should be distinguished from *peers* forming the overlay.

Adding new information. Initially, the map contains no density information and the root of its quadtree has no children. Thus, the local responsibility of the whole keyspace is given to the root, and a default density value[6] is assigned to the root. The triplet $(coord, rad, d)$ is then locally inserted in the quadtree following Algorithm 1.

The goal of the algorithm is to approximate the circular density area by a set of squares. The size of the squares may be variable, but in order to achieve a reasonable approximation accuracy, their size must be smaller than the circle. Therefore, the algorithm needs to find suitable squares in the quadtree and assign correct density values to each square. It explores the quadtree in search of leaves which squares 1) intersect the circle and 2) are smaller than the circle.

The algorithm progressively "zooms" on the square that contains the origin of the circle (loop from line 2 to 15). If the square that contains the origin is bigger than the circle, it is split (lines 3 to 4). Then, the value $coef$, which is the proportion of the surface occupied by the intersection with the circle, is computed for each of the current node's subsquares except the next subsquare on the path of the zooming process (lines 8 to 14). The influence of the circle's density on the updated density of the subsquare is proportional to $coef$ (line 10). If the child responsible for the subsquare is a leaf, its density value is simply updated (line 12). Otherwise, the value is propagated to its children (line 14). The propagation to the children is also performed by calculating the intersection with the circle for each of the descendants. The zooming process stops when the side of the square that contains the origin of the circle is smaller than the diameter of the circle.

Once the center of the approximation has been found, the algorithm ends by computing its new density (lines 16 to 22). This is done by calculating its intersection with the circle (lines 16 to 18), and propagating the value if the central square is not a leaf (line 22).

[6]This value is set to zero in our evaluations, but an approximation of the mean overlay density may help.

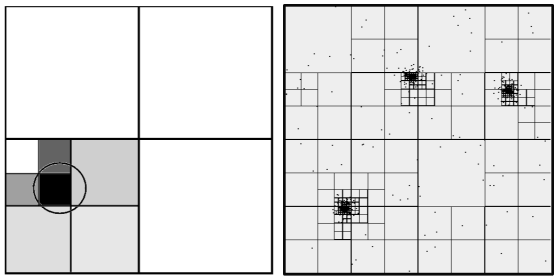

Figure 2. **Left:** After local execution of Algorithm 1 on a void map. (the greyscale represents the density, black being the highest). **Right:** Approximation of a keyspace with three density zones.

Algorithm 1: Upon the insertion of $(coord, rad, d)$ in the quadtree.

```
   Result: Information about the local density is incorporated to the map.
1  currentSq = rootOfQuadTree;
2  while currentSq.sideSize > 2 × rad do
3      if isLeaf (currentSq) then
4          currentSq.splitNode ();
5      next = currentSq.subSquareContaining (coord);
6      foreach child in currentSq.children do
7          if child is not next then
8              intersection = overlapCircleSquare (child,coord,rad);
9              coef = intersection / child.surface;
10             newDensity = coef× d +(1 − coef) × child.density;
11             if isLeaf (child) then
12                 child.density = newDensity;
13             else
14                 child.propagateToChildren (newDensity);
15     currentSq = next;
16 intersection = overlapCircleSquare (currentSq,coord,rad);
17 coef = intersection / currentSq.surface;
18 newDensity = coef× d +(1 − coef) × currentSq.density;
19 if isLeaf (currentSq) then
20     currentSq.density = newDensity;
21 else
22     currentSq.propagateToChildren (newDensity);
```

Exchanging information between peers. On each peer,

information about the density of a map region r is held by the subtree which root is responsible for the area that contains r. Thus, a peer a willing to share information about r with a peer b sends to b a message containing the corresponding subtree. Upon the receipt of the update message from a, b executes Algorithm 2 in order to merge the received subtree with its local quadtree.

First of all, b needs to locate the region of its map that is concerned with the update. Each node in the quadtree is in charge of its own region of the keyspace map, and two nodes cannot be responsible for the same region. Thus, a region referenced by a node of the tree may be identified by the location of its node in the tree. As b receives an update for a region of the map, it is able to locate the corresponding node in the local quadtree (lines 4 to 7). However, the node responsible for that region may not exist in the local quadtree. In that case, the merge algorithm splits the quadtree until the node is created (lines 5 to 6).

Then, the merging process may begin. Let u be the root of the subtree received in the update message, and l the local node in charge of that region on the map. There are four possibilities at this stage of the algorithm: 1) u and l are both leaves. In that case, the density value of u is simply assigned to l (line 10); 2) l is a leaf and u is not. Then the subtree of u is assigned to l (line 12); 3) u is a leaf and l is not. Then, the subtree of l is deleted (line 15) and the value of u, more up-to-date, is assigned to l; 4) none of the nodes are leaves: the merge algorithm is recursively called on each child of the nodes (line 18). At the end of the algorithm, the local map of b holds new information about the density in the region r of the keyspace. Algorithm 2 is designed to erase obsolete information (line 15), which is important since the density of the keyspace is likely to evolve over time.

Algorithm 2: Upon the receipt of a quadtree update from a distant peer.

```
   Result: The distant update has been merged with the local tree.
 1 currentSq = rootOfQuadTree;
 2 receivedRoot = root of the received update-subtree;
 3 mergeSubtree
 4     while not representSameRegion (currentSq,receivedRoot) do
 5         if isLeaf (currentSq) then
 6             currentSq.splitNode ();
 7         currentSq = currentSq.subSquareContaining (coord);
 8     if isLeaf (currentSq) then
 9         if isLeaf (receivedRoot) then
10             currentSq.density = receivedRoot.density;
11         else
12             currentSq.subtree = receivedRoot.subtree;
13     else
14         if isLeaf (receivedRoot) then
15             currentSq.deleteSubtree;
16         else
17             for childId in NW, NE, SW, SE do
18                 mergeSubtree (currentSq.child [childId ],
                   receivedRoot.child [childId ]);
19 end mergeSubtree
```

Means of propagation. We implemented two distributed mechanisms to propagate the subtree-updates among peers. First, we used a slightly modified gossip algorithm. An important property of our mechanism is the fact that peers that are semantically close to each other have extremely correlated maps. Thus, the updates between them would be nearly useless. For this reason, in our gossip protocol, a peer p locally assigns priorities to its overlay neighbors. The priority of a neighbor is inversely proportional to its graph distance to p. It means that a peer is likely to propagate updates only through its long-range links. Second, taking into account that the keyspace distribution follows data popularity, and is therefore likely to evolve over time, our mechanism should avoid to propagate outdated information about the distribution. Therefore, the data propagated to the neighbors during the gossip process is selected by novelty: recent information is propagated first.

Another possibility is to use join messages of arriving peers. As a peer j joins the overlay, it first connects to an entry point peer e that may be semantically located anywhere in the overlay. Semantic coordinates are then assigned to j according to some attribution mechanism. The coordinates of j are usually semantically far from e, and j has to reach its semantic neighbors through the overlay. A join request is routed by e to the semantic coordinates of j. At each step, it is possible to add some information about the density surrounding the current-hop peer. Each time the request reaches another peer on its route, the latter can benefit from the knowledge accumulated inside the request during the previous hops. It is rather an efficient way to propagate density knowledge. However, our evaluation shows that the amount of exchanged data increases with churn, while the increase is very moderate for the gossip propagation.

Using one of the mechanisms described above is necessary to assimilate changes in the key distribution that occur over time. On the other hand, a peer that just joined a fully bootstrapped overlay has no need to rebuild a full map from scratch. The map is relatively lightweight (see Section V) and can be recovered from an overlay neighbor during the join process.

IV. Drawing density-aware long-range links

Thanks to the collective use of the algorithms described in the previous section, peers progressively acquire an approximate map of the keyspace density distribution. This section describes how this knowledge may be used to build efficient long-range links in order to decrease the latency of the message routing.

Graph distance estimation. Thanks to the algorithms described in the previous section, each peer owns a density map of the bi-dimensional keyspace. This information allows it to locally *estimate* its graph distance to any coordinates of the keyspace. Namely, having the density distribution and the semantic distance, a peer is locally able to approximate the graph distance to the coordinates.

Let src be the coordinates of the peer, $dest$ the coordinates of the keyspace on which the estimation is performed, $d_{[src,dest]}$ the semantic distance and $GD_{[src,dest]}$ the graph

distance between them. If the key distribution is uniform all the way from src to $dest$, $GD_{[src,dest]}$ is roughly proportional to $d_{[src,dest]}$. More precisely, the semantic distance is equal to the number of hops to reach $dest$ in the graph multiplied by the mean euclidean distance of one hop multiplied by a shrinking constant k, *i.e.,* $d_{[src,dest]} = GD_{[src,dest]} \times meanHopDist \times k$.

The constant k is added because geometrically, the euclidean distance between src and $dist$ is shorter than the sum of the lengths of all the hops in the graph. The exact value of k depends on the topology of the graph, but can be empirically approximated. For instance, for the Delaunay graphs used in our simulations (see Section V), we have $k = 0.5$. We now need to calculate $meanHopDist$, the mean euclidean distance of one hop.

Assume n points uniformly distributed inside a square SQ of size $S = side \times side$. It is possible to approximate the mean euclidean distance between two points that are neighbors in the square. Let $meanDist_x$ and $meanDist_y$ be the projections of that mean distance on the x-axis and the y-axis. Since the nodes are uniformly distributed across SQ, we have $meanDist_x = meanDist_y = \frac{side}{n}$. That allows us to compute the mean distance between two neighbors in the square which is $meanHopDist_{SQ} = \sqrt{meanDist_x^2 + meanDist_y^2}$.

In case of heterogeneous distributions, regions crossed by the line $[src, dest]$ have different densities, so that the graph distance is no more proportional to euclidean distance. Therefore, the peer has to retrieve from the map information about the density distribution on the way from src to $dest$. The peer first determines the set $S = \{sq_1, sq_2, ..., sq_n\}$ of map square-zones that intersect the segment $[src, dest]$. Let $L = \{l_1, l_2, ..., l_n\}$ be the set of segments of $[src, dest]$ formed by its intersections with S. It is important to notice that:

- L forms a partition of $[src, dest]$. Therefore we may assume $GD_{[src,dest]} = \sum_{i=1}^{n} GD_{l_i}$.
- The quadtree contains one density-value per square, so the density inside each square is considered to be uniform. Therefore, $\forall i$, GD_{l_i} can be computed by the peer.

Thanks to the two previous assertions, the peer is able to approximate the graph distance *locally*.

The rewiring process. We use the log-partition algorithm described in Section II to achieve a d-harmonic distribution of long-range links. However, no random walks are performed. Instead, at each step of the algorithm, our system locally *estimates* the subset's population using the technique described above. To bootstrap the rewiring process, a peer p needs to find coordinates k in the keyspace that are estimated to be the farthest in terms of graph distance. This is done in order to reach as many peers as possible with the future long-range distribution[7].

[7]This procedure also gives an estimation of the graph diameter, and thus of the overlay size, which may be useful.

The localization of the farthest coordinates in terms of graph distance is not straightforward. Indeed, with heterogeneous distributions, the coordinates with the highest semantic distance are not necessarily the farthest in terms of graph distance if the keyspace has more than one dimension. As random uniform sampling of the map in search of the maximal distance is inefficient, another technique has to be employed. We propose to use the following property: if a node Y is the farthest from a node X in terms of graph distance, X and Y are responsible for opposite semantic values in at least one dimension of the keyspace.

Formally, let U be a n-dimensional borderless semantic keyspace, min_{dim} and max_{dim} the minimal and the maximal value of U for the dimension dim. Consider $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ the coordinates in U of two nodes X and Y of the topology supporting U.

Property. *If Y is the farthest node from X in terms of* graph distance*, then Y is responsible for coordinates $c = (c_1, c_2, ..., c_n)$ of the keyspace such that:* $\exists dim \in [1..n] : |x_{dim} - c_{dim}| = \frac{(max_{dim}+min_{dim})}{2}$

Proof:

1) Each dimension dim of U is borderless (*i.e.,* min_{dim} is semantically close to max_{dim}). Due to the modulo, the maximal semantic distance achievable in dim is $dmax_{dim} = \frac{(max_{dim}+min_{dim})}{2}$.
2) Let X and Y be two nodes and d_{dim} the value which is at a distance $dmax_{dim}$ from X in $dim \in [1..n]$ with $|d_{dim} - x_{dim}| = dmax_{dim}$. Let M be the farthest point from X belonging to (XY) in the direction of Y. Then, $\exists i$ such that $M = (m_1, ..., d_i, ..., m_n)$. Assume that Y is not responsible for M. Therefore, M is located farther from X than any coordinates of (XY) owned by Y. Thus, whether a) M is not owned by any node or b) M is owned by another node Z. The supposition a) is impossible because all the coordinates have a root. The supposition b) implies that thanks to the greedy routing, the graph distance to reach Z from X is higher than the graph distance to reach Y *i.e.,* Y is not the farthest from X in terms of graph distance. By 1) and the contrapositive of 2), the property is correct. □

In our case, the two dimensions have the same size, therefore $min_1 = min_2 = min$, $max_1 = max_2 = max$ and $dmax = \frac{(max+min)}{2}$. Let P be a peer of the overlay with coordinates $p = (p_1, p_2)$, and $S = \{x \in U : max(|p_1 - x_1|, |p_2 - x_2|) = dmax\}$. The property implies that $\exists M \in S$ such that M is owned by the peer that is the farthest from P in the graph. The coordinates of S form a square centered on P with a side of size $dmax$.

The peer P uses a Monte Carlo method [23] on its density-map to uniformly sample a set of coordinates that belong to the square formed by S. Then, the samples are ordered by their estimated graph distances to the peer. The sample M for which the estimated graph distance is the highest is supposed to be owned by the farthest peer. A shortcut-request is routed to these coordinates in the overlay, and the peer responsible

for it becomes the longest link of the routing table.

Then, P determines by dichotomy the coordinates MID that belong to the segment $[PM]$ for which the estimated graph distance is the half of the estimated graph distance to M[8]. A long-range request is routed to MID and the process is repeated with the segment $[P, MID]$. Similarly to Oscar, the process ends when one of the close neighbors of P is reached (or when the wanted number of links has been created).

If the process has ended without achieving the wanted number of long-range links, a random member of S is selected, and the process is repeated until the wanted number of shortcuts is reached. At the end of the algorithm, the peer has the wanted number of long-range links that are distributed following an approximation of the d-harmonic distribution.

Putting aside the cost of the density map maintenance protocols, the rewiring process is extremely lightweight: all estimations are done locally. Therefore it requires only one routing process per long-range link. Since the number of links to add follows $log(n)$ and the routing process is still efficient during the rewiring of a peer, the overall cost in number of messages is expected to grow following $log^2(n)$.

V. Evaluations

This section presents a detailed evaluation of DONUT. We first describe the evaluation environment. Then, we compare the long-range rewiring process of DONUT to state-of-the-art rewiring techniques. Finally, the behavior of the map maintenance algorithms is studied. All the systems have been evaluated in PeerSim, a widespread discrete event simulator [24].

A. Evaluation environment

To perform our evaluations we simulate a bi-dimensional keyspace. This choice was made because it allows multiple-range queries, and some applications using such keyspaces are already well identified [9], [17]. The keyspace forms a square, each side of which is bound to the opposite side to ensure the modulo. We choose to use a Delaunay triangulation topology to support the keyspace. This choice is believed to be relevant because 1) the greedy routing algorithm is proven to work for all Delaunay based topologies [13]; 2) such topologies are already used by overlay designers [15]; 3) the triangulation is generalizable to higher dimensions.

The environment of our simulation is dynamic: peers join and leave the overlay. We use both real and synthetic traces. Our synthetic churn model follows an exponential law for disconnections and a Poisson law for joins. This model is widely used in the literature to simulate the availability of electronic components. The parameters of the two laws are bind to ensure a roughly constant overlay size. In our evaluations with that model, we vary the mean session time of peers from 10 minutes to 6 hours. The mean session times we observed in the peer-to-peer churn traces is bound by these values.

Our semantic keyspace is formed by three high density zones that contain 90% of the overall peer population. The distribution of peers inside each density zone follows a Zipf-law. The rest of the population is uniformly distributed between the hotspots. Such distributions seem to be rather common for existing keyspaces (see Section II). For instance, this density distribution is shown to be comparable to the population distribution of Second Life, a popular MMOG [25].

In real life, the dynamicity of the environment impacts the characteristics of the keyspace. When a peer joins the overlay, it is inserted in the keyspace with new coordinates. The coordinates may be defined by the overlay *e.g.,* the peer joins the most overloaded region, helping to support the load. It is also possible that the coordinates are user-defined (in a distributed game, the player chooses to join a particular region). As the application load varies in time, and popularity zones may evolve, the density of the semantic keyspace changes over time. In order to study the adaptive capabilities of DONUT, the coordinates of all the density hotspots periodically change. The mean session time also impacts the keyspace: once the hotspot coordinates have changed, it determines the new hotspot growth-speed.

Finally, to make our simulations more realistic, we use:

- A matrix containing real-latency traces.
- Real-system churn traces.

We use real latencies that were measured by Madhyastha et al. between 2500 hosts spread over the Internet [26]. We use churn traces collected from several existing distributed systems, such as Overnet [27] or Skype [1]. Traces of desktop personal computer usage were also injected [28].

Since the latency matrix has 2500 entries, except for Figure 4, all synthetic evaluations were performed with an overlay size of 2500. We simulate *one week* of activity. The gossip map-maintenance protocol period is set to ten minutes, and is allowed to propagate 60 Kbytes to three direct neighbors per period. For the propagation by lookups, each lookup message may contain 10 Kbytes of map data. The rewiring protocol occurs once per hour, and the coordinates of all hotspots change once per 24 hours.

B. The rewiring process evaluation

We compare DONUT to several techniques of long-range link construction, namely:

- **Random approach:** n coordinates are selected uniformly at random in the keyspace.
- **Uniform Kleinberg approach:** the log-partition algorithm described in Section II is used, and the keyspace is assumed to be uniform. Therefore, the graph distances are supposed to be proportional to semantic distances.
- **Near Optimal approach:** the peer locally has a real-time copy of the current topology. The estimated graph distance equals the real graph-distance.

[8] *I.e.,* the segment $[PM]$ is divided in two semantically equal parts. If the graph distance of the semantic middle is less than half of the graph distance to M, the second interval is taken, and so on.

- **Oscar:** algorithm described in [11] (see Section II). However, the original Oscar algorithm is designed for one-dimensional keyspaces. Therefore, we have extended Oscar to our bi-dimensional keyspace.

Each time a peer joins the overlay, the join message is routed to the semantic position the peer has chosen. We measure the performance of that routing process[9]. The evaluation shows that DONUT outperforms all the strategies (except the Near Optimal one) by at least 20%.

A first important result concerns the Oscar strategy, which exhibits poor results in our bi-dimensional keyspace. For each measurement, Oscar had only slightly better results than the random strategy. In particular, Figure 4 shows that the number of hops during the greedy routing linearly grows with the overlay size. The result is a bit surprising, because the system performed well in one-dimensional keyspaces [11]. We believe that the problem comes from random walks used by Oscar to sample the keyspace. This sampling technique is successful only on graphs that have good expansion properties [10]. Flaxman showed that the graph used by Kleinberg (a bi-dimensional lattice with d-harmonic shortcuts) is not an expander. We believe that Delaunay graphs with d-harmonic shortcuts are not expanders. This results in non uniform sampling of the keyspace, skewing the estimation. Indeed, when the random walks are replaced by a cheat mechanism that uniformly picks random nodes from the node-set of the simulator, Oscar shows much better performance.

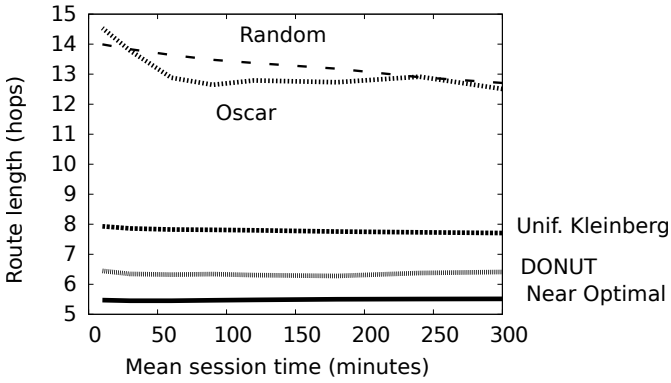

Figure 3. Mean route length as a function of the mean session time, lower is better. Lower session time means higher churn.

Figure 3 describes the evolution of the route length while varying the mean session time. We can see that the churn rate has no significant influence on the Near Optimal, Uniform Kleinberg and DONUT approaches. For the Near Optimal technique, it happens because each peer locally has a real-time graph of the topology and is therefore perfectly aware of the topology for any churn rate. The Uniform Kleinberg approach assumes the uniformity of the keyspace and thus behaves the same way despite the churn rate increases. On the other hand, the fact that the efficiency of DONUT does not decrease shows that the map adapts itself on time and is again accurate when the rewiring is performed. Figure 4

[9]Both latency and number of hops.

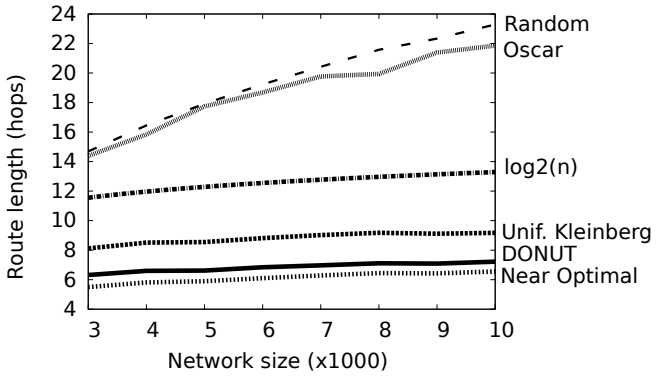

Figure 4. Scalability test: Mean route length in function of the overlay size.

shows that except the random strategy and Oscar, all the rewiring processes scale well. The increase of the path-length seems to be logarithmic for DONUT as well as for the Uniform Kleinberg and the Near Optimal approaches. Like in the other evaluation results, DONUT is the closest to the Near Optimal approach, showing that most of the graph distance estimations performed by DONUT are accurate. The bad scalability of the random approach comes from the fact that the coordinates of future long-range links are chosen uniformly across the keyspace. The links are thus likely to "miss" most of the density hotspots, increasing the inefficiency of the approach. The positive impact of

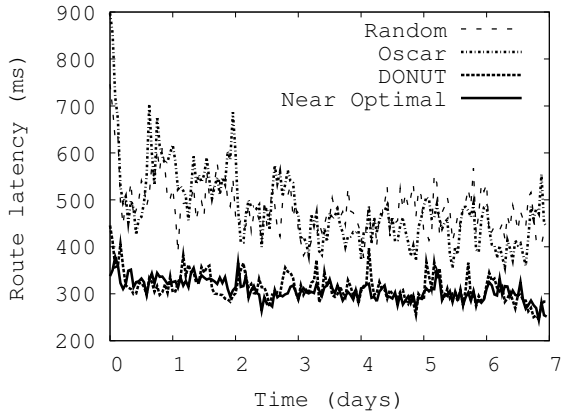

Figure 5. Evolution of routing latency with the Overnet trace.

the density map on the rewiring process is confirmed while using real traces. Figure 5 shows the evolution of the mean routing latency while using churn traces from Overnet. The graph distance estimation provided by DONUT is almost as efficient as the use of a real-time graph. One may notice that DONUT is sometimes even better than the Near Optimal strategy. However, the difference is very slight, and is probably due to the standard deviation of the measurements. Figure 6 recapitulates the average latencies and the standard deviations for all the strategies while injecting the Overnet churn trace.

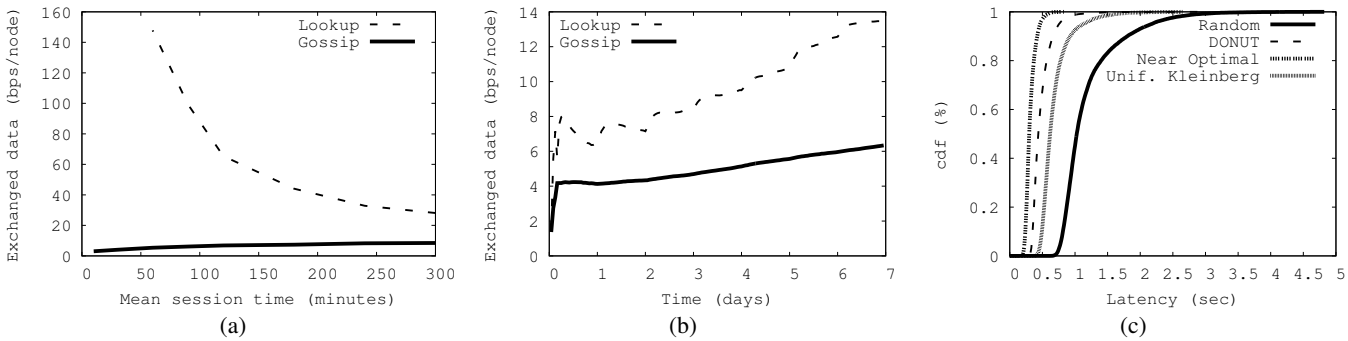

Figure 7. Measurements of density map propagation. (a): Sent data per node in function of the mean session time; (b): Evolution of the network load with the Overnet trace; (c): Cumulative distribution function (cdf) of mean routing latencies for an overlay of 2500 nodes.

Strategy	Average Latency	Standard Deviation
Random	464.5	41.1
Oscar	438.2	44.6
Unif. Kleinberg	370.2	31.8
DONUT	300.7	27.3
Near Optimal	295.6	18.9

Figure 6. Average latency and standard deviation (in milliseconds) for all strategies with the Overnet trace and the real latency matrix

C. Global map propagation evaluation

The first part of the evaluation showed that our density map was helpful to the rewiring of the overlay. However, the construction of a *global* map may seem to be a costly process. Therefore, it is important to measure the network cost of DONUT.

The coordinates and the size of each region are deterministically defined by the path in the tree from that region to the root. One integer value is sufficient to store that information for most of the paths. In addition to that, each leaf of the tree stores a double value representing the density of its region. For an 2500-node overlay and the density distribution described above, the average quadtree contains 77 nodes and 232 leaves. As a regular quadtree node is stored on 4 bytes, and a leaf is stored on 8 bytes the average quadtree size is about 2.2 Kbytes only.

This size is comparable to other maintenance meta-information such as for instance, bloom filters in PAST [6]. Moreover, a peer is able to control the fuzziness of the map in order to optimize its size. A quadtree node is responsible for a region formed by the union of its children's regions. If the densities of the subregions are equivalent, the children may be suppressed and their mean density value may be affected to their parent.

Figure 7.a shows the network load of DONUT maintenance for the two different propagation strategies described in Section III. We can see that the Lookup Propagation strategy significantly increases as the session time decreases. This is due to the fact that the information is propagated inside join messages, which number increases with churn. Moreover, at each step of the join-message routing, the forwarding node adds its local information even if it has been already propagated before. Therefore, the lookup messages are always full of density information. For a mean session time equal to 30 minutes, the network load of the propagation is of 0.6 Kbytes per second per node. This cost is still affordable, and the approach does not require the implementation of a dedicated dissemination protocol. For these reasons, the lookup propagation may interest designers of systems with mean session times below 30 minutes.

On the other hand, the modified gossip algorithm has a near-constant network load because: 1) the protocol is not related to join messages and 2) peers only propagate *new* information: the gossip messages most of the time contain much less information than the maximal allowed size. Thanks to that, regardless of the churn rate, the network cost of gossiping the density map is below 10 bytes per second per node, which is very low.

Figure 7.b shows the evolution of the network load while using the Overnet churn trace. Here again, the Lookup propagation uses more bandwidth than the gossip algorithm. Moreover, the lookup propagation strategy induces important variations of the network load over time, which may be harmful to the overlay. Results of evaluations with other real churn traces are omitted due to a lack of space but exhibit the same characteristics.

Figure 7.c has been realized by using the real latency traces. It shows the cumulative distribution function of routing latencies. As expected, the Near Optimal strategy exhibits the best distribution characteristics. This is due to the fact that *all* peers have the same accurate local graph. DONUT significantly outperforms the other techniques, approaching the Near Optimal strategy[10]. The mean routing latency of DONUT is less than $500ms$ for 67% of the peers. On the other hand, only 0.006% have the same mean latency with the random approach. The Uniform Kleinberg approach behaves slightly better than random: 11% of the peers have less than a $500ms$ mean routing latency, which is still much less than DONUT. 98% of the peers implementing DONUT have a mean routing latency less or equal to $1s$, while less than a

[10]In DONUT case, the lookup propagation and the gossip algorithm are both equally efficient, so Figure 7.c makes no distinction between the two strategies.

half (45%) exhibit the same characteristics for the random strategy. These values show that the propagation of the density map is efficient for a large population subset. Thanks to that, and regardless to their position in the keyspace, most of the peers are able to build efficient shortcuts in the overlay.

VI. Conclusion

A growing number of distributed applications require a support for efficient range querying. In range query overlays, the uniformity of resource-key distribution is not guaranteed and studies show that existing distributions are heterogeneous. Furthermore, the distributions evolve under the churn impact. Providing an efficient routing service in such conditions is difficult, because peers need to be locally aware of the topology to accurately choose shortcuts in the overlay. Existing solutions do no monitoring of the topology evolution and are only able *react* when the routing performance drop.

We propose DONUT, a mechanism that builds a map of the peer distribution and uses the map to create efficient shortcuts in the overlay. DONUT's map adapts itself to the evolution of the distribution, providing to each peer a mean to estimate graph distance to any coordinates of the keyspace. Our evaluations show that: 1) Obtained shortcuts offer a scalable routing process; 2) shortcuts built with the map increase routing performance by more than 20% compared to Uniform Kleinberg, thus approaching the near optimal algorithm; 3) the density maps are extremely lightweight: the mean map size is of 2.2 Kbytes only for 2500 peers and efficient map propagation consumes less than 10 bps of bandwidth on each peer.

As for the DONUT's perspectives, we believe that the keyspace density map can be useful to other important distributed system mechanisms such as reactive load balancing, network size estimation, optimized routing algorithms, or global system monitoring. These applications will be developed in our future work.

References

[1] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *IPTPS*, February 2006.

[2] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS*, ser. LNCS, P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds., vol. 2429. Springer, 2002, pp. 53–65.

[3] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *IPTPS*, ser. LNCS, M. Castro and R. van Renesse, Eds., vol. 3640. Springer, 2005, pp. 205–216.

[4] J. M. Kleinberg, "The small-world phenomenon: an algorithmic perspective," in *STOC*, 2000, pp. 163–170.

[5] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM*, 2001, pp. 161–172.

[6] P. Druschel and A. I. T. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *HotOS*. IEEE Computer Society, 2001, pp. 75–80.

[7] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP*, 2001, pp. 202–215.

[8] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi, "A peer-to-peer framework for caching range queries," in *ICDE*. IEEE Computer Society, 2004, pp. 165–176.

[9] A. R. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games," in *NSDI*. USENIX, 2006.

[10] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," in *SIGCOMM*, R. Yavatkar, E. W. Zegura, and J. Rexford, Eds. ACM, 2004, pp. 353–366.

[11] S. Girdzijauskas, A. Datta, and K. Aberer, "Structured overlay for heterogeneous environments: Design and evaluation of oscar," *TAAS*, vol. 5, no. 1, 2010.

[12] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, May 2001, ch. Gnutella, pp. 94–122.

[13] P. Bose and P. Morin, "Online routing in triangulations," *SIAM J. Comput.*, vol. 33, no. 4, pp. 937–951, 2004.

[14] O. Beaumont, A.-M. Kermarrec, and E. Riviere, "Peer to peer multidimensional overlays: Approximating complex structures," in *OPODIS*, ser. LNCS, E. Tovar, P. Tsigas, and H. Fouchal, Eds., vol. 4878. Springer, 2007, pp. 315–328.

[15] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: a scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[16] F. Bonnet, A.-M. Kermarrec, and M. Raynal, "Small-world networks: From theoretical bounds to practical systems," in *OPODIS*, ser. LNCS, E. Tovar, P. Tsigas, and H. Fouchal, Eds., vol. 4878. Springer, 2007, pp. 372–385.

[17] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world," in *PDPTA*, H. R. Arabnia and Y. Mun, Eds. CSREA Press, 2003, pp. 262–268.

[18] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, pp. 440–442, June 1998. [Online]. Available: http://dx.doi.org/10.1038/30918

[19] S. Milgram, "The small world problem," *Psychology Today*, vol. 1, p. 61, 1967.

[20] M. Ripeanu, I. T. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *CoRR*, vol. cs.DC/0209028, 2002.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM*, 1999, pp. 126–134.

[22] S. Girdzijauskas, A. Datta, and K. Aberer., "On small world graphs in non-uniformly distributed key spaces," in *ICDE Workshops*, 2005, p. 1187.

[23] R. Motwani and P. Raghavan, "Randomized algorithms," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Ed. CRC Press, 1997, pp. 141–161.

[24] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," http://peersim.sourceforge.net/.

[25] S. Legtchenko, S. Monnet, and G. Thomas, "Blue banana: resilience to avatar mobility in distributed mmogs," *DSN*, vol. 0, pp. 171–180, 2010.

[26] H. V. Madhyastha, T. E. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A structural approach to latency prediction," in *Internet Measurement Conference*, J. M. Almeida, V. A. F. Almeida, and P. Barford, Eds. ACM, 2006, pp. 99–104.

[27] R. Bhagwan, S. Savage, and G. Voelker, "Understanding Availability," in *Proceedings of IPTPS'03*, 2003.

[28] W. Bolosky, J. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs," in *Proceedings of SIGMETRICS*, 2000.

[29] E. Tovar, P. Tsigas, and H. Fouchal, Eds., *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*, ser. LNCS, vol. 4878. Springer, 2007.