Fuzzynet: Zero-maintenance Ringless Overlay*

Sarunas Girdzijauskas‡, Wojciech Galuba‡, Vasilios Darlagiannis‡
Anwitaman Datta†, Karl Aberer‡

‡Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

†Nanyang Technological University (NTU), Singapore

Abstract

Many structured overlay networks rely on a ring invariant as a core network connectivity element. The responsibility ranges of the participating peers and navigability principles (greedy routing) heavily depend on the ring structure. For correctness guarantees, each node needs to eagerly maintain its immediate neighboring links - the ring invariant. However, the ring maintenance is an expensive task and it may not even be possible to maintain the ring invariant continuously under high churn, particularly as the network size grows. Furthermore, routing anomalies in the network, peers behind firewalls and Network Address Translators (NATs) create non-transitivity effects, which inevitably lead to the violation of the ring invariant. We argue that reliance on the ring structure is a serious impediment for real life deployment and scalability of structured overlays. In this paper we propose an overlay called Fuzzynet, which does not rely on the ring invariant, yet have all the functionalities of structured overlays. Fuzzynet takes the idea of lazy overlay maintenance further by dropping any explicit connectivity and data maintenance requirement, relying merely on the actions performed when new Fuzzynet peers join the network. We show that with sufficient amount of neighbors ($O(\log N)$, comparable to traditional structured overlays), even under high churn, data can be retrieved in Fuzzynet w.h.p. We validate our novel design principles by simulations as well as PlanetLab experiments and compare it with ring based overlays.

*The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935. The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

1 Introduction

Structured overlays, e.g., Distributed Hash Tables (DHTs) provide essential indexing and resource discovering in distributed information systems. Typically, structured overlays are based on enhanced rings, meshes, hypercubes, etc., leveraging on the topological properties of such geometric structures. The ring topology is arguably the simplest and most popular structure used in various overlays [5, 14, 24, 29, 32]. In ring based overlays, it is necessary and sufficient to set correctly the successor and the predecessor of each node for correct routing, while additional (long range) links are used to enhance routing efficiency.

Under churn (peer membership dynamics), the ring is both a blessing and a curse. On the one hand, an intact ring is sufficient to guarantee correct routing. Hence, historically, all existing structured overlays have de facto considered it necessary. We argue that it is not only *un*necessary, but also relying on such a ring invariant leads to some undesirable consequences. In certain cases, the existing greedy-routing mechanisms cannot deal with even a single fault/break in the ring on the routing path. On the other hand, in a dynamic environment where peer lifetime is few minutes for the majority of them, the ring is susceptible to continuous breakages. This in turn incurs high maintenance cost, and despite whatever high maintenance cost, there is at no point any absolute guarantee that the ring is indeed intact. The larger the number of peers, the more likely it is that the ring invariant is violated. This is a serious impediment for scalability and deployment of structured overlays.

Moreover, another well-known challenging issue for the ring invariant is posed by the non-transitive connectivity and the routing anomalies in the overlay networks. It is quite common in real-life networks that some pairs of alive peers can not directly communicate to each other (e.g. between two firewalled peers); however, it is possible for them to communicate indirectly through a third peer. As it has been shown in [7], such non-transitive connectivity may misdirect nodes to wrongly set ring successors to keys, thus leading to violation of the ring invariant and disrupting the

overlay's functional correctness.

The effect of unreachable nodes has been studied in depth by several researchers. Kong et al. [20] investigated the percolation effect in structured peer-to-peer (P2P) systems such as Chord [32] and Symphony [24] and measured the size of the reachable network component under failures. In the experiments the authors expose the drawbacks of these systems, specifically showing that up to 4% of the nodes are not reachable (where the network size is 10^6 peers) even though they belong to the same connected component. Mislove et al [26] found routing anomalies in 9% of PlanetLab [17] peer pairs, where the peers could not establish direct connection among themselves. Wang et al [34] measured two real peer-to-peer systems and found that even up to 36% of the participating peers were residing behind firewalls and Network Address Translators (NATs), which in many cases made direct communication between these peers impossible. These results show that the inevitable deficiency in the direct communication between any two peers prevents sustaining the ring invariant in real networks. Therefore, it seems that despite the great maintenance cost, in reality structured overlay networks have huge challenges meeting the assumed system invariance.

In this paper, a different approach called *Fuzzynet* is investigated, which circumvents the need for a ring and the associated problems like non-transitivity and costly maintenance. By introducing the Fuzzynet technique we set a base for a completely *lazy* design of P2P systems where the only maintenance action is taken upon peers joining the network. Fuzzynet is based on the connectivity principles of navigable Small-World networks [18]. It does not require the ring structure, yet have all the functionalities of contemporary structured overlay networks. Fuzzynet peers can "mimic" the ring-behavior by contacting the immediate key-neighbors through the neighbor cluster with high probability by exploiting Small-World clusterization. More particularly, the suggested relaxed structure of Fuzzynet has the following differences compared to tightly structured DHTs: i) No explicit ring maintenance. ii) Peers are not deterministically responsible for a particular key section but probabilistically. iii) Data keys are disseminated and replicated in the vicinity of the targeted key. Fuzzynet peers develop their neighbors according to a policy for optimal routing at the joining phase and this effort helps older peers update their stale connections. As it is shown later, this suffices assuming stable churn rate.

While Fuzzynet is based on loose connectivity and is much more relaxed in its peer-to-key bindings, it is not an unstructured overlay. The peer keys and the stored data keys are highly correlated. The lookup messages in Fuzzynet are never flooded but greedily routed to the targeted area based on the data keys. Even though our system's performance guarantees are probabilistic rather than deterministic, we show that with sufficient amount of neighbors ($O(\log N)$, comparable to traditional overlays), even under high churn the data can be retrieved w.h.p. from our system. In contrast, traditional overlays which rely on a ring provide a deterministic guarantee subject to the condition that the ring invariant is met. However, in reality, this invariant is impossible to meet continuously. As a consequence, systems relying on the ring invariant have poorer performance over time on the average than a probabilistic system, as we observe from the experiments (cf. Section 5). Moreover, our suggested solution does not depend on any key distribution, giving Fuzzynet the flexibility to achieve further desired system properties such as load-balancing.

Thus, in contrast to the related literature which tries to improve the ring maintenance mechanisms, we take a complementary approach, where we want to ensure functional correctness (of querying and new data insertions) even in case the ring invariant is violated. Whenever the ring invariant is met, our approach has no message overheads compared to the traditional approaches given similar data replication factor (which is anyway needed for fault tolerance). Therefore, the mechanism can be integrated to work seamlessly in a ring-based P2P network, while avoiding the non-transitivity problems and obviating the need for any aggressive and expensive ring self-stabilization. For the purpose of overall efficiency, a low-cost background self-stabilization mechanism may however be employed.

The paper is organized as follows. In Section 2 we describe the basic concepts and the design of Fuzzynet, and in Section 3 we present and discuss the relevant algorithms. In Section 4 we give a theoretical analysis of the approach. In Section 5 we validate our design based on simulations and experiments with a Java based Fuzzynet prototype implementation on the PlanetLab [17] testbed. We discuss related systems in Section 6 before drawing our conclusions in Section 7.

2 Ringless Overlay

2.1 The Need For the Ring Structure

To begin the quest of "removing the ring" first we have to understand why one needs the ring in the first place. There are plenty of reasons why the ring is an attractive design solution for distributed indexing systems. We will discuss the most important of them.

Navigability. First of all, the ring[1] makes the small world easy to navigate, i.e. using decentralized memoryless greedy routing algorithm. The introduction of such routing

[1] We can generalize the ring to a "kleinbergian" lattice [18] or any other exact, peer key-dependent structure like hypercubes [30], butterfly networks [23], etc.

technique necessitates the ring structure to assure the correctness of the routing algorithm. Without the ring structure the query messages would not have any guarantees of reaching the desired peer since there will be no assurance for forwarding, i.e., an intermediate peer might not have a "closer" link to the target, thus failing the query.

Responsibility. Secondly, and even more importantly for data-oriented P2P systems, the ring structure provides clear responsibility space for every peer. E.g., in Chord a peer p is responsible for all data items which hash into the range $\left[p(key), p_{successor}(key)\right)$. With such a knowledge every peer certainly knows which identifer range it is responsible for and which query messages have already "reached the target" and do not need to be forwarded further. The storing/routing/answering decisions can be made because of the certainty that there is no other peer between two successive ring neighbors. Since these storing/routing decisions are basic and essential for any data-oriented P2P system, the ring has to be maintained eagerly (e.g. periodically). Eager maintenance is required even if the churn rate in the network is high and the ring connections (which were established with high cost) have never been used before they are dropped.

Easy Construction of Long-Range Links. Thirdly, having the ring as always-in-place concept, it is relatively easy to use it as a bootstrap building block of the long-range links of the P2P system. E.g. Skip graphs with their "multi-dimensional" rings [4, 15], or the hop count technique [19].

Although the ring invariant is a very strong requirement, because of the aforementioned advantages most of the structured overlays employ this idea and impose the ring structure in their approaches (e.g. [3, 5, 10, 24, 32]).

However, as discussed in the Introduction, even with a high maintenance cost it is practically impossible to meet the ring invariant assuming realistic network conditions, where abundance of participating peers reside behind firewalls and NATs contributing to the frequent routing anomalies making the direct communication between some ring links impossible. Hence, we take a completely different standpoint to design a concept which would not require such a strong assumption as the ring invariant.

2.2 Fuzzynet Concepts

In order to be able to drop the ring invariant we need to address the aforementioned functional requirements which make the ring an attractive solution in the P2P community.

Firstly, Fuzzynet drops the requirement for every peer having a predefined deterministic responsibility range on the identifier space. Instead, we use a probabilistic responsibility approach, where a data item will be likely to be stored on a peer whose key on the identifier space is close to the hash value of that data item (data key).

Secondly, we employ a data replication concept in Fuzzynet, by disseminating the data replicas in the vicinity of the data position on the identifier space. Since P2P overlays (Small-World networks) exhibit a high clusterisation effect, the data dissemination in the vicinity of the desired position can be performed with relatively low effort. Such dissemination of data replicas does not actually rise the requirements for our system, since all the realistic systems (which use the ring structure) employ replication for fault tolerance and persistence anyway. An useful consequence of such data replication in Fuzzynet is the fact that a simple greedy routing query will find one of the replicas w.h.p. given sufficient network connectivity.

To make the Fuzzynet concept work we need to be able to construct a navigable overlay (Small-World network) without the help of the ring. For that we can use some of the existing approaches, e.g. Oscar's sampling technique [13] for skewed key distributions or [8] for uniform key distributions.

In the following we will discuss in more detail the above described principles of Fuzzynet, which can be generalized as two types of routing: routing for lookup (read) and routing for storing or publishing (write).

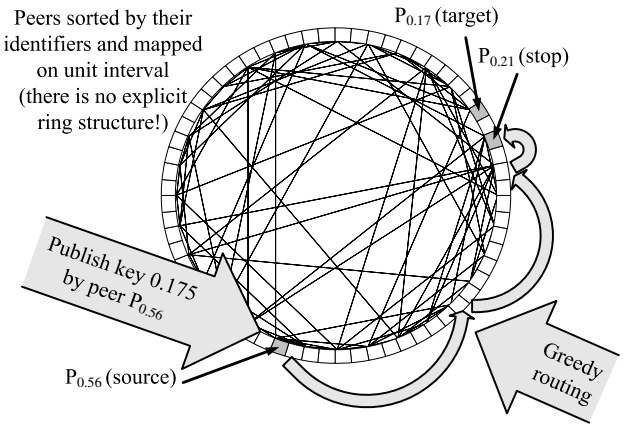
2.2.1 Lookup (Read)

Lookup routing will employ a greedy routing algorithm, where messages will be forwarded every time minimizing the distance to the target. The routing terminates if a data item D, which was looked-up is found. However, since there are no ring links and no predefined responsibility ranges, a peer might end up in a situation where it does not have any links which would lead the query closer to the target, nor it holds the requested data. In such a case the lookup query would terminate unsuccessfully. Nevertheless, we will prove later in the analysis and show with the experiments that with realistic parameters w.h.p. data is found if it was published before (e.g. in the networks with $O(\log N)$ degree and typical peer replication cost).

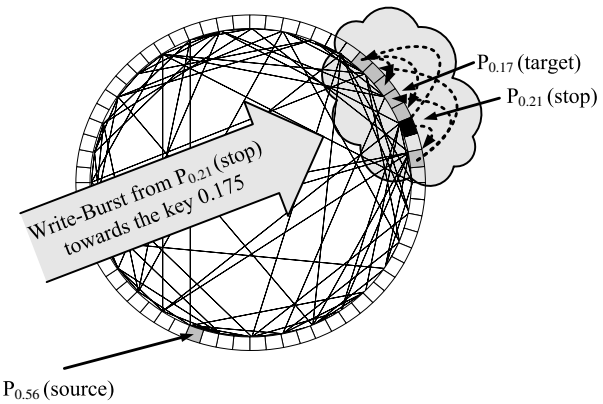
2.2.2 Publish (Write)

The high guarantees for the lookup lie in the exploitation of a particular Small-World property, namely the clusterization property, during the data writing phase.

The write operation is performed in two stages and stores data D on r peers (replicas) in the vicinity of the data key $D(key)$. The first stage is similar to the lookup (read) phase and uses greedy routing to find one of the peers which are close enough to the data key $D(key)$. Once the write operation reaches the vicinity of the targeted key location, the data is seeded in the nearby peers by the self-avoiding multicast (a controlled "Write-Burst"). The underlying idea is to use the clusterization property of the network and to reach



(a) Greedy-Approach. Routing from the originator peer ($P_{0.56}$) to the *greedy-closest* peer ($P_{0.21}$) where the greedy approach towards the target key 0.175 (*actual-closest* peer $P_{0.17}$) is no further possible.



(b) Write-Burst. The *greedy-closest* peer ($P_{0.21}$) seeds the replicas in the cluster vicinity of the key 0.175 using the Write-Burst.

Figure 1. Schematic example of two Publish (Write) phases: Greedy-Approach and Write-Burst.

as many peers as possible in the $D(key)$ vicinity. The multicast has two parameters - fn (fanout) and $depth$. A peer contacts its fn closest neighbors to $D(key)$ and requests to store the data item D as well as to continue the multicast process with reduced $depth$. The multicast avoids the peers which have been visited (already store data D) and terminates when $depth$ reaches zero. Data D is seeded (stored) on all the peers reached by the multicast-burst. It will be shown later in Section 5 that it is sufficient to have fanout $fn = 2$ and $depth = 3$ for successful storage and retrieval in a system with $O(\log N)$ network degree. An example of the publish (write) procedure is given in Figure 1(a) and Figure 1(b) followed by the example of the successive lookup (read) operation in Figure 2.

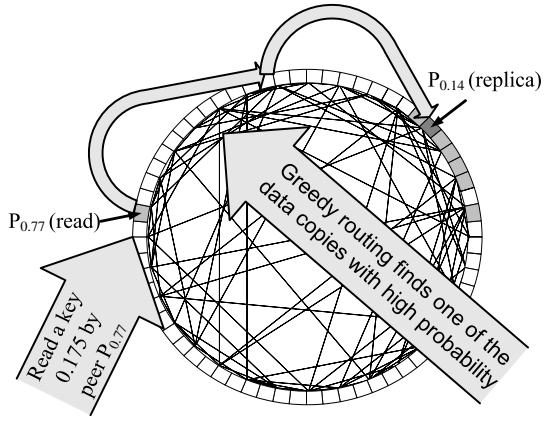


Figure 2. After writing the data in the vicinity of the key 0.175, the lookup (read) from any node will have very high chance finding at least one of the data replicas.

In contrast to "classical" decentralized data-oriented systems, the replicas in our probabilistic overlay do not need to be globally aware of each other. Although the write procedure is more complex than read (lookup), there are no messages wasted, i.e., only peers which will be storing the data are contacted.

By exploiting the clusterization property, the Write-Bursts avoid the necessity of the ring and circumvent the non-transitivity problems. In a way, the bursting technique finds the bypasses to the "would-be" ring-neighbors by choosing second or third best neighbors and relying on the fact that in a Small-World network, peers in the same vicinity are highly connected. Instead of keeping the ring structure alive periodically (as in the classical P2P systems) the write procedure in the probabilistic overlay "imitates" the ring behavior only for the storage, whereas the read (lookup) does not actually need the ring if the replication factor is sufficiently large. For all practical purposes, the minimal amount of replication used by current systems purely for the purpose of fault tolerance appear to be sufficient enough.

2.2.3 Updates

Fuzzynet treats data updates in the same way as data insertions, i.e. using the above described "Publish" concept. The updates are made when a new (updated) value D' of the existing data D is published on the same key $D(key)$. During publishing, the Write-Burst seeds the updated data items in the vicinity of the key $D(key)$, overwriting the existing old data. Although the replica placement in the vicinity of the D(key) is non-deterministic, their high degree of clusterization typically ensures that a suitable implicit write quorum is found to overwrite the older version. We assume that

standard mechanisms for version control are employed by the application layer.

2.2.4 Maintenance

The basic idea of our approach is to ensure a sufficient performance of the system with no explicit maintenance. We do not have any assumptions on how peers leave, i.e. we deal similarly both on departures and failures. In such a way we can provide much stronger guarantees that our system will be functional given any circumstances. In the environment where the churn rate is stable, i.e. similar amount of peers join and leave/crash, it is not necessary to perform any maintenance of the overlay except the one triggered by the arrival of peers. The churn itself is typically enough to keep the system functioning, i.e. the newcomers with the fresh links will compensate the lost connectivity due to crashed peers [19].

Similarly, the churn can keep the residing data alive when the newcomers replicate and republish the data if deemed necessary (cf. Section 3.2.3). Every time a new peer joins the network, it checks all the data which is similar to its identifier key. The newcomer can decide to refresh the data by performing a new write if it notices that the replication factor is too small, i.e. a peer does not see enough replicas.

3 Algorithms

Here we formally describe the algorithms used in our system. Fuzzynet is a general technique that can be implemented on any Small-World topology which does not require the ring for constructing the network (e.g. [8]). In this work, for establishing the connectivity of our system we use the Oscar [13] overlay construction algorithms, which do not require the ring entity and can cope with non-uniform (skewed) key spaces. In the following we will briefly describe Oscar working principles; however, for detailed algorithms and analysis please refer to our previous works [13, 14].

3.1 Building the Small-World Network Using Oscar Algorithms

It is known that for building a routing efficient network with skewed key spaces one needs to know the probability density function of peer identifiers over the identifier space [12]. One of the simplest ways of doing that is to randomly sample the network and get an approximation of the key distribution, e.g. Mercury [5]. However, the real-world distributions can be totally arbitrary and the only sufficient approximation of the distribution would be gathering in a sample set the complete set of values which, of course, does not scale. In our previous work [13] we have shown that using such a technique Mercury fails to build routing efficient networks if arbitrary distribution functions occur. Moreover, we have also shown that it is not necessary to know the distribution function over the entire identifier space with uniform "resolution" – it is sufficient to "learn" well the distribution for only some regions of the identifier space while leaving other regions vaguely explored, making it the base idea of Oscar algorithms.

Oscar uses this intuition in order to build its routing network in a simple and efficient manner. An Oscar node u with a key (identifier) $u(id)$ has to partition the identifier space into logarithmic partitions $A_1, A_2, ..A_{\log_2 N}$, where N is the number of peers in the system. Each border between neighboring partitions is determined by a median value of the peer identifiers in the logarithmically decreasing peer populations, i.e. the border between A_1 and A_2 will be the median m_1 of the peer identifiers from the whole peer population $\mathcal{P}$, the border between A_2 and A_3 will be the median m_2 of the identifiers from the subpopulation $\mathcal{P} \setminus A_1$, etc. In general the border value between A_i and A_{i+1} will be the median m_i of peer identifiers from the subpopulation $\mathcal{P} \setminus B_i$, where $B_i = \cup_{j=1}^{i-1} A_j$. Ideally the first partition A_1 has to contain $\frac{1}{2}$ of the initial population, A_2 has to contain $\frac{1}{4}$ and so on. Since in practice it is not possible to exactly know the precise members of all the partitions, an Oscar node has to approximate the key range for each partition. For finding the median values an Oscar node has to uniformly sample each subpopulation B_i and determine the current median m_i from the acquired sample set. To sample the subsets of the population B_i the Oscar nodes use random walkers which do not visit nodes with identifiers that do not belong to the current population B_i. The technique yields very good results in practice even with very low sample sizes.

Applying the results from [12] we formulate the basis of Oscar's technique – the long-range link acquiring procedure: *each peer u first chooses uniformly at random one logarithmic partition A_i and then within that partition uniformly at random one peer v. This peer v will become a long-range neighbor of u.* Regardless the complexity of the distribution function it is sufficient to sample only $O(\log N)$ medians, hence the Oscar sampling technique is always scalable. The number of long-range links in Oscar is not restricted and can be assigned individually according to the needs of a particular peer, as long as there exist at least one such link per peer. It has been proven that in the worst case the search in Oscar network will be $O(\log^2 N)$.

3.2 Fuzzynet Data Management Algorithms

Having established a Small-World network we can employ the Fuzzynet algorithms to enable successful data storage and retrieval without the presence of a ring. Here we will describe the core data management algorithms of our ringless data-oriented overlay.

3.2.1 Write-Burst Algorithm

Algorithm 1 Publish (Write) algorithm $publish(data2store)$

1: $[burstPeer] = greedyRoute(D(key))$
2: $writeBurst(burstPeer, data2store, D(key), fanout, depth, \oslash)$

To store a data item D a peer would initiate a two-phase publish algorithm (Algorithm 1). In the first phase it would look-up the closest peer ($currentPeer$) to the data key $D(key)$ it can find with greedy routing algorithm, and once found, the "Write-Burst" (Algorithm 2) would be initiated at the $currentPeer$. The algorithm would contact the closest neighbors to $D(key)$ (number of neighbors defined by $fanout$) from the $currentPeer$. Once the closest neighbors are reached the algorithm would store the data D on the visited peers and recursively continue contacting the closest peers, until the maximum allowed $depth$ is reached (i.e. $depth = 0$). The algorithm is made to be self-avoiding, i.e. once the peer is visited - it is skipped in later on. With "Write-Burst" algorithm we exploit the clusterization property of small world networks and recursively visit as many peers in the vicinity of $D(key)$ as possible.

3.2.2 Lookup (read) Algorithm

The lookup or read algorithm is fundamentally the same as a traditional greedy routing algorithm. When a lookup request is issued on $D(key)$, the query travels greedily towards the target examining at each peer whether it stores a copy of D. Once terminated (no more possibility to get closer to the target) the algorithm analyzes the collected information about the data D and with high probability (cf. Section 4) a replica of D is found.

3.2.3 Peer Join Algorithm

As discussed earlier, in this work the join algorithm uses the original Oscar techniques to wire the network, i.e. to ensure the connectivity and the desired Small-World properties. Here we will discuss only the second stage of the join process, i.e. the data management after the network connectivity is established.

Algorithm 2 Write-Burst algorithm $[visited] = writeBurst(p, D, D(key), fanout, depth, visited)$

1: $depth = depth - 1$
2: **if** $depth \geq 0$ **then**
3: $visited = visited \cup p$
4: store D on p
5: $notVisitedNeighbors = = getNeighbors(p) \setminus visited$
6: $fanoutCounter = = min(fanout, notVisitedNeighbors)$
7: **while** $fanoutCounter > 0$ **do**
8: $CloseLink = chooseClosestToTheKey (notVisitedNeighbors, D(key))$
9: $[visited] = writeBurst(CloseLink, D, D(key), fanout, depth, visited)$
10: $fanoutCounter = fanoutCounter - 1$
11: $notVisitedNeighbors = = notVisitedNeighbors \setminus visited.$
12: **end while**
13: **end if**

Once a peer p joins the network and establishes its connections it needs to copy the data which is stored in the vicinity of the peer's key $p(key)$ in the key space. For this reason the newcomer peer p performs a Write-Burst algorithm, but instead of writing the data, it collects all the "visible" neighbors in the vicinity of the joining peer's key $p(key)$. Once peer p collects the data from the neighborhood peers it analyzes it (Algorithm 3, line 3). Peer p copies (becomes a replica) of all the data whose keys are close enough to its own key $p(key)$ given the existing replication rate induced by $fanout$ and $depth$ parameters (Algorithm 3, line 5). The estimation whether a data item D belongs to the peer's vicinity is easy even for the non-uniform key distributions, since a peer knows in advance the boundaries of the logarithmic partitions $A_1, A_2, .., A_{\log N}$ from the overlay building process (cf. Section 3.1) and can estimate the number of peers residing between $D(key)$ and $p(key)$.

Similarly, a joining peer p could estimate how many replicas of a particular data item D should be visible, taking into account the key of the current data item $D(key)$ and peer's p key $p(key)$. In case the amount of data items is lower than some predefined threshold, peer p can initiate the write algorithm to reinsert data item D. In such a way, if peers leave/crash and arrive independently, it is ensured that the data item will not be lost once it was written in the P2P system.

Algorithm 3 data acquisition algorithm upon join $[targetPeer] = join(p)$

1: $clusterNeighbors = \emptyset$
2: $[clusterNeighbors] =$
$= writeBurst(p, \oslash, p(key), fanout, depth, \oslash)$
3: **for** $\forall data\ D \in clusterNeighbors$ **do**
4: **if** $estimateIfDataInVicinity(p(key),$
$D(Key), fanout, depth)$ **then**
5: store D
6: **end if**
7: $estimatedNumberOfDataItems =$
$= estimateData(p(key),$
$D(key), fanout, depth)$
8: **if** $estimatedNumberOfDataItems <$
$< tresholdNumber$ **then**
9: $publish(D)$
10: **end if**
11: **end for**

4 Analysis

In this section we will show the computational lower bounds for the success probability to retrieve a data item D once it was stored on our system. We will investigate the case of Write-Burst for publishing data, where the parameters are the following: $fanout = 2$ and $depth = 3$ and the size of the routing tables at each peer is $\rho(p) > \log N$. As our simulations suggest, these are the smallest values with which the system performs reasonably well (cf. Figure 4) having relatively low average network degree. Although with smaller Write-Burst values the success rate might still be high enough, there will be much fewer replica copies in the system, thus increasing the risk to loose all of them in case of unexpected increase in churn. Therefore, with the aforementioned parameters we will establish the general lower bound of the success probability for acquiring a stored data item in our system.

We will calculate the success probability in three steps. In the first step we will calculate with what probability a write/read message can reach the vicinity of the data key if the ring connectivity is not enforced (i.e. when no more greedy routing is possible). In the second step we will calculate the probability that the Write-Burst will populate the immediate neighbors of the Write-Burst originator. And in the third step we will combine the two and will calculate what is the general success probability for a read message to reach a peer which holds a written data item.

4.1 Step 1. Routing Without Ring-Links

Here we will calculate the probability P_ρ of a message to be delivered from a originator peer p_o to the target peer p_t using only the existing Small-World network links without the ring connectivity assumption. We assume the worst case scenario when the distance $d(p_o, p_t)$ is maximal, (e.g. equal to 0.5 in the setting of the unit interval where the links between the peers are bi-directional and the peer keys are distributed uniformly). Let us divide the space between peer p_o and peer p_t into logarithmic partitions $B_1, B_2, .., B_i$ where B_1 partition contains the closest $\frac{1}{2}$ of the peers residing between $p_o(id)$ and $p_t(id)$, B_2 - the further $\frac{1}{4}$ of the peers and so on. If the ring links were present and $\rho(p) > \log N$ the message from the peer p_o to the peer p_t will have to hop on average $\log N$ peers, at each peer diminishing the distance to the target logarithmically, i.e. by at least one logarithmic partition B_j [12]. However, without the ring links, starting from the second hop there exist a non zero probability that there will be no links pointing to the peers which would allow greedy approach to the target, i.e. the ring neighbor link is missing. The probability of not-getting closer is relatively small in the first hops of the query, but it grows significantly when the message approaches the target. It is because the remaining logarithmic partitions become smaller and smaller thus a query can find fewer and fewer links which point into the remaining partitions. Let us denote with P^j_{hop} the probability that a message will be able to approach greedily to the target at each hop j. In Oscar (and in Small-World networks in general) the probability for a peer to choose a neighbor belonging to any of the logarithmic partitions $A_1, A_2, .., A_{\log N}$ is the same, and equal to $\frac{1}{\log N}$. Since A and B partitions are symmetric, the probability P^j_{greedy} that being at jth hop a query can find at least one link pointing into one of the partitions $B_j, B_{j+1}, .., B_{\log N}$ is equal to $1 - (\frac{j-1}{log_2 N})^{\rho(p)}$. Since we assume the worst case scenario, i.e. the peer p_o is the furthest form the peer p_t, the P^1_{hop} will be equal to 1, i.e. any link from p_o could greedily traverse towards the target. From the second hop, there exists a possibility not to be able to advance towards the target (no ring connectivity) therefore for every next j hop when $j > 1$ we have to calculate P^j_{hop} recursively. In general we obtain $P^j_{hop} = P^{j-1}_{hop} \cdot (1 - P^j_{greedy}) = P^{j-1}_{hop} \cdot (1 - (\frac{j-1}{log_2 N})^{\rho(p)})$.

4.2 Step 2. Replication by Write-Burst

When a "write" message for the data item D reaches a peer p_t from which no more greedy routing can be performed, the Write-Burst algorithm is evoked to replicate the data in the surrounding area of $p_t(id)$. The peer p_t can immediately store the data, so it will be populated with data D with probability 1; however, the neighboring peers do not have 100% guarantees to get populated. Here we will investigated the probabilities for the two closest peers form the left (p_{t-1},p_{t-2}) and from the right (p_{t+1},p_{t+2}) of the initiator peer p_t to get populated by the new data. According

to the kleinbergian Small-World network construction principles the probability P'_1 that a peer will have a direct long range link to one of its immediate neighbors is $P'_1 = \frac{1}{\log_2 N}$. Similarly, the probability P'_2 that a peer will have a direct long range link to one of its immediate neighbors' neighbor (2^{nd} order neighbor) is $P'_2 = \frac{1}{2\log_2 N}$. The probability P''_1 that a peer will be connected to one of its immediate neighbors by two hops is at least $P''_1 \geq P'_1 \cdot P''_2$. Likewise, the probability P''_2 that a peer will be connected by two hops to one of its immediate neighbors' neighbor (2^{nd} order neighbor) is $P''_2 \geq P'^2_1$. Therefore the probability of a peer p_t to reach and store data on the peer p_{t-1} or the peer p_{t+1} is $P_1 = 1-(1-P'_1)(1-P''_1)$. Similarly, the probability of a peer p_t to reach and store data on the peer p_{t-2} or the peer p_{t+2} is $P_2 = 1-(1-P'_2)(1-P''_2)$.

Let us assume a peer p'_t is the closest peer to the data identifier which has to be written. The probability that the write message will stop on that peer is $P_{hop}^{\log N}$. Similarly, the probability that the message will stop on $p'_{t\pm1}$ and on $p'_{t\pm2}$ is $P_{hop}^{\log N-1}$ and $P_{hop}^{\log N-2}$, respectively. In order for the data to be written on the node p'_t, the Write-Burst algorithm had to start either in the p'_t node itself or in its neighborhood. We will calculate what is the the probability $P_{p'_t}$ that a data item D was written on node p'_t if the write algorithm stopped on p'_t immediate neighbors ($p'_{t\pm1}$) or one of its neighbor neighbors ($p'_{t\pm2}$). The probability that a Write-Burst algorithm started at $p'_{t\pm1}$ node and the data item D was written on the node p'_t is $P_{hop}^{\log N-1} \cdot P_1$. Likewise, the probability that the Write-Burst algorithm started at $p'_{t\pm2}$ node and the data item D was written on the node p'_t is $P_{hop}^{\log N-2} \cdot P_2$. Therefore, $P_{p'_t} \geq P_{hop}^{\log N} + P_{hop}^{\log N-1} \cdot P_1 + P_{hop}^{\log N-2} \cdot P_2$. Similarly, we can calculate the probabilities that the data item D was written on the nodes $p'_{t\pm1}$ and $p'_{t\pm1}$ having the write burst algorithm started in the vicinity of these nodes. Therefore, $P_{p'_{t\pm1}} \geq P_{hop}^{\log N-2} + 0.5 \cdot P_{hop}^{\log N} \cdot P_1 + 0.5 \cdot P_{hop}^{\log N-2} \cdot P_1 + 0.5 \cdot P_{hop}^{\log N-1} \cdot P_2$ and $P_{p'_{t\pm2}} \geq P_{hop}^{\log N-2} + 0.5 \cdot P_{hop}^{\log N-1} \cdot P_1 + 0.5 \cdot P_{hop}^{\log N} \cdot P_2$.

4.3 Step 3. Lower Bound on Success Probability (Worst Case Scenario)

Having the probabilities that the data item D was written in the vicinity of the identifier of D', we can calculate what will be the lower bound of the success probability P_{succ} if a "read" message will be issued at some peer in the system. Since we know what are the probabilities for a greedy routing algorithm to advance towards the target without using the ring-links we can calculate that P_{succ} is at least $P_{succ} \geq P_{p'_t} + P_{p'_{t\pm1}} \cdot P_1 + P_{p'_{t\pm2}} \cdot P_2$.

In Figure 3 we can see the plot of the lower bound success probabilities with different sizes of networks and network's degrees. The above result suggests that given an average network degree of $2\log N$, regardless the network size N, the queries in Fuzzynet will be successful w.h.p. even in the worst case scenario. In practice, the success rate in Fuzzynet is even higher (cf. Figure 4).

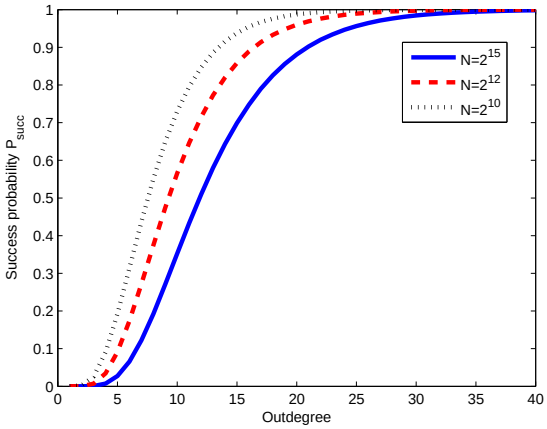


Figure 3. Lower bound on success probability.

5 Experimental Results

5.1 Simulations

In our simulations we have experimented with Fuzzynet technique built on top of Small-World networks. We investigated what are the probabilistic guarantees to retrieve the data which was stored on the ringless overlays and also compared our approach to ring-based Small-World networks. We have simulated the network environments with routing anomalies where the peers behind firewalls and NATs were not allowed to establish a link directly with each other. Our experiments were carried out on various network connectivity cases, mainly networks with different node degrees. All the experiments were performed including peer churn as described next.

In the first part of the simulations we have performed tests with a network of 10000 peers and a predefined average node degree. For each case of the average degree we have performed separate experiments on the dynamic network with stable churn rate - i.e. one peer joins and one peer leaves (fails) at each time slot. Leaving and failing were considered as identical operations, since no action is taken upon a "graceful" leave. The correctness of the system relies only on the actions taken upon new peers joining the network: establishing new links and replicating data from the surrounding areas. We have modeled the behavior of the networks for over 20000 time slots. Upon the 1st time

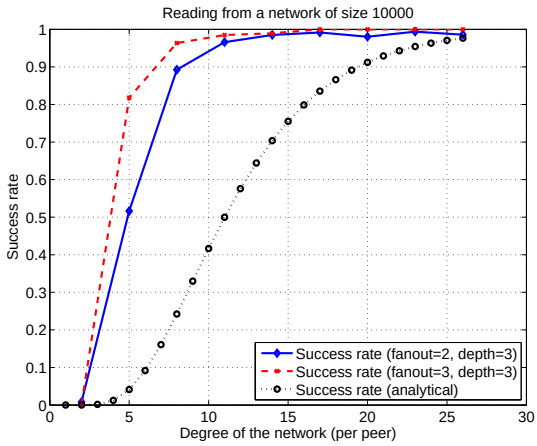


Figure 4. Query success rate in Fuzzynet under churn.

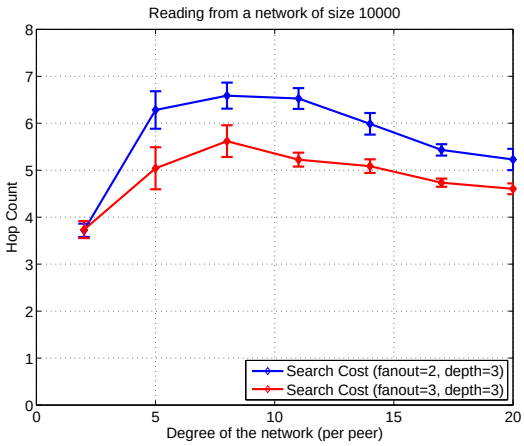


Figure 5. Total search cost of both: successful and unsuccessful queries (the error bars represent the standard deviation from the average search cost).

slot 100 unique data items were inserted into each peer with random keys. During the experiments we have captured the snapshots of the network after every 500 time slots. For each snapshot of the network we have performed a *read* query from every peer on the identifiers of the previously written data items. The average success rate and the average search cost (path length) of the queries were measured. Figure 4 shows the average query success rates given different fanouts and depths of the Write-Burst algorithm and various average degrees of the networks. We observe that with a relatively low average degree (14 links per node) the success rate rises up to almost 100%. In Figure 5 we show the average search cost for looking up the data.

We experimented with various rates of peer departures superceding the rate of new peers joining, as a result of which the network shrank (i.e., non-equilibrium scenarios). We studied the system's performance until it approximately shrank to half its original size, i.e. 5000 peers from the original 10000. We varied the shrinking rates from relatively slow ones, taking 5000 time slots to halve the network, up to very rapid ones, lasting only 5 time slots until the network reached half of its original size. Fuzzynet proved to be pretty resilient against even such drastic peer population changes. With the average network degree of 20 and with Write-Burst parameters $fanout = 2$ and $depth = 3$, the rate at which the network shrank had a very weak influence on query success rate, which stayed above 96%. Notice that tolerating such a huge membership change within a short interval is essentially equivalent to having a correlated failure of the corresponding number of peers, and hence we conclude from these experiments that Fuzzynet can deal with a massive correlated failure, even without any repair operations, because it does not need the ring invariant for functional correctness.

In the second part of the simulations we have compared the Fuzzynet technique to ring-based approaches performing under the faulty environments. As a ring-based overlay we have implemented Symphony [24] algorithms. To simulate the effects of peers under firewalls we have labeled some of the peers as "firewalled" peers. During the lifetime of the networks we have forbidden the direct communication among any two labeled peers [34]. We have also simulated sporadic routing anomalies by forbidding a communication between randomly selected fraction of existing links [26]. In the experiments we have monotonously increased the fraction of firewalled peers and the probability of routing anomalies, such that the total link failure rate was increasing from 0 to 0.3. We have simulated churn as in the first part of the simulations and measured the performance of the networks every 500 time slots. The results in Figure 6 show that even with high link failure rates the Fuzzynet technique with parameters higher than $fanout = 2$ and $depth = 2$, has much higher success rate than a ring-based approach. With lower values, there are not enough replicas to sustain the data under churn, hence the queries fail to find some of the previously inserted data items, since they disappear from the network. However, with higher values of the these parameters, no data is lost, given the existing replication. Figure 7 depicts the data persistence history - an average amount of data replicas residing in the network at each time slot (10000 peers, 20 links on average at each peer).

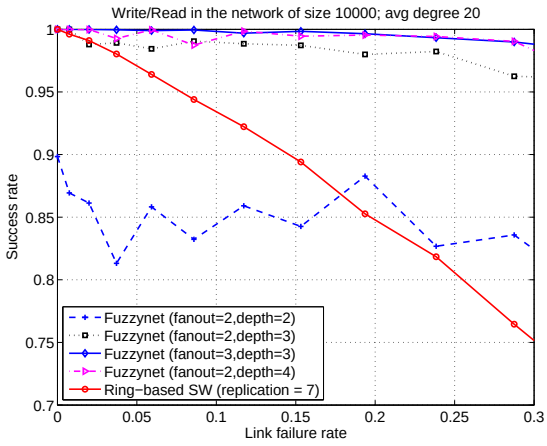


Figure 6. Fuzzynet success rate as compared to a Ring-Based overlay.

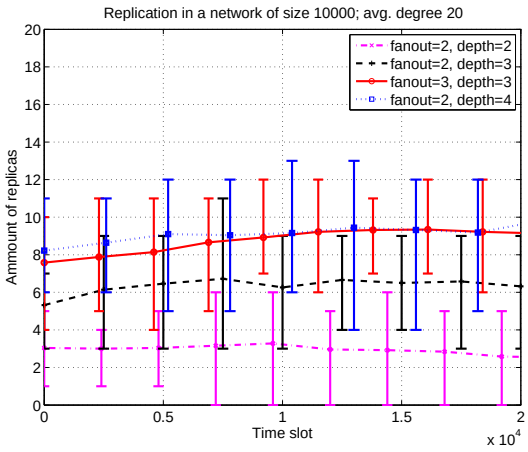


Figure 7. Data persistence over time. The solid lines represent the average amount of replicas per data item at a particular time slot (the error bars represent the 5th and the 95th percentiles of the replica size distribution).

5.2 Experiments on PlanetLab

We have also implemented Fuzzynet using the ProtoPeer [16] toolkit. The system is deployed on 330 PlanetLab nodes, all communication between the overlay neighbors is done via TCP and all other communication is done via UDP. For the first 5 minutes we bootstrap the system into a Small-World ring-based topology with Chord-like connectivity. At the 5 minute mark all peers insert 50 random and unique key-value pairs into the overlay. At the 10 minute mark all peers start to periodically lookup a randomly chosen key out of the $330 * 50$ inserted. We measure the system-wide failure rate of lookups. A lookup fails if while being routed no greedy next hop is possible but the current peer does not contain the desired key. We also measure the average path a lookup took together for both successful and failed lookups. In addition, for each key we count how many times it was replicated, which is summarized as the average, 5^{th} and 95^{th} percentiles of the number of replicas.

We run the experiment in two different setups (Table 1): i) A Chord-like write/read algorithms, where there is only one peer responsible for a particular data item and the replication of that data (Chord) and ii) Fuzzynet write/read algorithms with a replication induced by different $depth$ and $fanout$ values (cf. Section 3.2.1). The same experiments are repeated for a network with simulated firewalled peers. Two firewalled (NAT) peers cannot be overlay neighbors since they cannot directly communicate between each other. We set the fraction of firewalled peers to 36%, following the results of the study [34] which measured the number of peers behind firewalls and NATs in large-scale public P2P systems.

In our experiments we have observed that, indeed, due to network anomalies even in the absence of NAT a fraction of ring links are missing. The results show that in a real live network deployment the missing ring links cause a considerable number of unsuccessful data insertion operations and, consequently, failed lookups in Chord's case (2.15%). Fuzzynet's write/read algorithms, however, lowers the failure rate by two orders of magnitude (0.01% in the case of D3 F2) only with an average of 6 replicas. Under realistic assumptions on firewalled hosts [34], the Chord topology is even more disrupted with 5.2% of failed lookups. The Fuzzynet approach lowers the loss 10-fold down to 0.47%.

Some of the WriteBurst branches stall and time out, PlanetLab is notorious for its unpredictable delays [27]. We have not implemented any acknowledgements or retries, still our replication scheme is robust under the loss and delay conditions of PlanetLab and enough replicas are created to significantly reduce lookup failures.

	avg # hops	failed lookups	# Fuzzynet replicas		
			avg	5^{th}	95^{th}
Chord	4.27	2.15%	-	-	-
Fuzzynet D2 F2	3.70	0.15%	4.38	3	5
Fuzzynet D3 F2	3.55	0.03%	6.20	5	7
Fuzzynet D14 F1	3.17	0.01%	14.26	8	15
Chord NAT	4.44	5.20%	-	-	-
Fuzzynet D2 F2 NAT	3.79	1.81%	4.64	4	6
Fuzzynet D3 F2 NAT	3.65	0.47%	6.72	5	9
Fuzzynet D14 F1 NAT	3.79	1.16%	4.64	4	6

Table 1. Summary of the PlanetLab results. "D" and "F" represent different depth and fanout values (cf. Section 3.2.1)

6 Related Systems

The vast majority of structured overlays base their topologies and routing techniques on exact, peer key-dependent core structures like rings [24, 32], trees [1, 25], de Bruijn graphs [9], hypercubes [30], butterfly networks [23], etc. As discussed in the motivation part - all of them to a very high extent lack the flexibility of choosing the neighbors on the close neighborhood level, what makes the maintenance more complicated. Many works were devoted only to tackle the problem of maintaining the exact structure (e.g. rings) under churn and various stabilization algorithms were developed to keep the core structures alive [2, 21, 22, 28, 31]. On the other side, there are unstructured and semi-structured systems [6, 11, 33] based on loose topology which require low maintenance, but are rather inefficient in terms of bandwidth consumption and are suffering from low query recall.

The seminal semi-structured system Freenet [6] requires only loose topology and low maintenance cost; however, it has no guarantees that the existing data can be retrieved even in the functioning network. In Fuzzynet data items are placed using the Small-World clusterization, so, even if they are not placed deterministically, it is easy to perform an "update" unlike in Freenet where data items are more widely spread in the overlay.

Another semi-structured system Yappers [11] trades-off between Gnutella-like flooding and DHT routing. In contrast to Fuzzynet's greedy routing - the lookups in Yappers are performed in a broadcast-like fashion. Effectively Yappers is only an improved version of Gnutella network, though it floods smaller fraction of peers than Gnutella itself.

The recently proposed BubbleStorm [33] uses "bublecast" - a data dissemination and querying strategy based on random walks with flooding over random multigraphs. BubleStrom is designed in such a way that the "data bubble" and the "query bubble" are very likely to intersect. Because of its unstructured nature BubbleStorm requires low maintenance cost; however, the exhaustive flooding-based querying is far too costly compared to Fuzzynet.

To the best of our knowledge Fuzzynet is the first structured overlay based on a loose connectivity, which while providing good recall guarantee (in fact, under real networking conditions it is better than the current ring based DHTs) while not requiring any exact-structure maintenance, thus drastically saving on maintenance overheads at a marginal overhead for overlay operations like data read and write.

7 Conclusions

In this paper we presented the Fuzzynet technique which allows building structured ringless overlays without requiring any explicit maintenance under churn. Unlike traditional ring-based overlays, Fuzzynet can successfully cope with non-transitivity problems and routing anomalies in realistic networks. Our technique can be employed on loose network topologies, permitting to avoid the eager maintenance strategies of the "heavy", peer key-dependent structures as e.g. the rings. We show that despite bearing a loose topology, Fuzzynet can still perform efficient publishing (write) and greedy lookups (read) with probabilistic guarantees and surpass ring-based overlays like Chord and Symphony in faulty environments as encountered in real life [34]. We have analytically calculated the lower performance bounds of Fuzzynet and evaluated it with simulations as well as with implementation and deployment of our system on PlanetLab.

We believe that Fuzzynet is ideal for high churn environments, and will successfully fill in the gap between the bandwidth wasting unstructured and high maintenance cost classical-structured overlays that rely on the integrity of the ring, and nevertheless perform worse under real life churn conditions than our probabilistic system based on fuzzy data placement.

As a future work we plan to exploit the properties of the restriction-free Fuzzynet technique beyond the maintenance cost reduction. The same flexible technique has a potential to address other load-balancing issues, particularly query-load balancing, which can be tackled by changing the replication rate for each individual data item. Also Fuzzynet can possibly address trust and reputation issues by adapting the "Write-Burst"-like mechanisms to serve as a peer voting technique in an asynchronous environment.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[2] D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *In 17th ACM Symposium on Parallelism in Algorithms and Architectures(SPAA 2005),Las Vegas, NV, USA*, 2005.

[3] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *PODC2004*, 2004.

[4] J. Aspnes and G. Shah. Skip graphs. In *SODA*, 2003.

[5] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *ACM SIGCOMM, Portland, USA*, 2004.

[6] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001.

[7] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. Non-transitive connectivity and dhts. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.

[8] W. Galuba and K. Aberer. Generic emergent overlays in arbitrary peer identifier spaces. In *2nd International Workshop on Self-Organizing Systems (IWSOS 2007)*, volume 4725, pages 88–102, 2007.

[9] E. Ganesan and D. K. Pradhan. Wormhole Routing in De Bruijn Networks and Hyper-DeBruijn Networks. In *ISCAS*, 2003.

[10] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB*, 2004.

[11] P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *INFOCOM'03, San Francisco, USA*, 2003.

[12] S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. In *NetDB2005, Tokyo, Japan*, 2005.

[13] S. Girdzijauskas, A. Datta, and K. Aberer. Oscar: Small-world overlay for realistic key distributions. In *DBISP2P 2006, Seoul, Korea*, 2006.

[14] S. Girdzijauskas, A. Datta, and K. Aberer. Oscar: A Data-Oriented Overlay For Heterogeneous Environments. In *ICDE 2007*, 2007.

[15] N. J. A. Harvey, Jones, M. B., S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS'03, Seattle, WA*, March 2003.

[16] http://protopeer.epfl.ch. Protopeer.

[17] http://www.planet lab.org/. Planetlab.

[18] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[19] F. Klemm, S. Girdzijauskas, J.-Y. Le Boudec, and K. Aberer. On Routing in Distributed Hash Tables. In *The Seventh IEEE International Conference on Peer-to-Peer Computing*, 2007.

[20] J. Kong and V. Roychowdhury. Price of structured routing and its mitigation in p2p systems under churn. In *P2P'07, Galway, Ireland*, 2007.

[21] X. Li, J.Misra, and G. Plaxton. Active and concurrent topology maintenance. In *In the 18th Annual Conference on Distributed Computing (DISC)*, 2004.

[22] D. Liben-Nowell, H. Balakrishnan, and D. R. Karger. Analysis of the evolution of peer-to-peer systems. In *PODC2002, New York, USA*, 2002.

[23] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[24] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *4th USENIX Symposium on Internet Technologies and Systems, USITS*, 2003.

[25] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[26] A. Mislove, A. Post, A. Haeberlen, and P. Druschel. Experiences in building and operating epost, a reliable peer-to-peer application. In *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 147–159, New York, NY, USA, 2006. ACM.

[27] S. Rhea, B. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of opendht on planetlab, 2005.

[28] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. Technical Report Technical Report UCB//CSD-03-1299. The University of California, Berkeley, Univ. Paris-Sud, 2003.

[29] 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), Heidelberg, Germany*, 2001.

[30] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup – hypercubes, ontologies and efficient search on p2p networks. In *Workshop on Agents and P2P Computing, 2002.*, 2002.

[31] A. Shaker and D. S. Reeves. Self-stabilizing structured ring topology p2p systems. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, 2005.

[32] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.

[33] W. W. Terpstra, C. Leng, and A. P. Buchmann. Bubblestorm: Resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM'07, Kyoto, Japan*, 2007.

[34] W. Wang, H. Chang, A. Zeitoun, and S. Jamin. Characterizing guarded hosts in peer-to-peer file sharing systems. In *In Proceedings of IEEE Global Communications Conference*, Global Internet and Next Generation Networks, 2004.