A Self-Stabilizing Locality-Aware Peer-to-Peer Network Combining Random Networks, Search Trees, and DHTs

Thomas Janson
University of Freiburg, Germany
janson@informatik.uni-freiburg.de

Peter Mahlmann
University of Paderborn, Germany
mahlmann@upb.de

Christian Schindelhauer
University of Freiburg, Germany
schindel@informatik.uni-freiburg.de

***Abstract*—We present 3nuts, a self-stabilizing peer-to-peer (p2p) network supporting range queries and adapting the overlay structure to the underlying physical network. 3nuts combines concepts of structured and unstructured p2p networks to overcome their individual shortcomings while keeping their strengths. This is achieved by combining self maintaining random networks for robustness, a search tree to allow range queries, and DHTs for load balancing. Simple handshake operations with provable guarantees are used for maintenance and self-stabilization. Efficiency of load balancing, fast data access, and robustness are proven by rigorous analysis.**

***Index Terms*—p2p, random graphs, robustness, self-stabilization, locality, search trees**

I. Introduction

Peer-to-peer (p2p) networks have become very popular for the exchange of resources (e.g. data). In contrast to client server architectures, the nodes (peers) of a p2p network have symmetrical functionality and each peer acts as server, client, and router at the same time. This property bears the potential of excellent failure resilience, since there is no single point of failure and the impact of individual failures may be less than in other architectures. Measurement studies of real world p2p networks reveal that these underlie high churn rates [1]: Peers frequently join and leave without prior notice. Thus, robustness is a key to realize the actual potential of p2p in applications and it is reasonable to choose a simple network structure that is easy to maintain, keeps the network functional under churn, and allows to recover quickly from degenerate states.

P2p networks can be divided into unstructured and structured networks according to their topology. Unstructured networks are characterized by the lack of constraints on data placement and topology. Each peer maintains its own local index and may select arbitrary other peers as neighbors resulting in a random graph like topology which is easy to maintain and provably robust [2]. For data discovery each local index has to be queried separately. So, complex queries such as range queries can be implemented easily, but have to be performed by broadcasts imposing a lot of network traffic or by random walks [3] which reduce traffic, but massively increase search latency. Structured networks overcome this problem by strictly controlling data placement and evolution of the topology. Often distributed hash tables (DHT) are used for data placement [4], [5], [6]. DHTs map data and peers into a virtual space M via hashing and assign data to the peer which is closest in M, for example. So data stored at the same peer is usually completely unrelated and queries are limited to exact match queries. The topology is controlled to allow efficient routing and often emulates well known static networks (cf. Fig. 1). Generally, structured p2p networks are considered to be harder to maintain under churn and less robust than unstructured networks due to the extra overhead to control topology and data placement [7]. DHTs constitute an important step towards scalable p2p networks. Yet, scalability is bought dearly by neglecting *locality* during data placement, resulting in the limitation to exact match queries.

We combine concepts of unstructured and structured p2p networks to overcome their individual shortcomings while keeping their strengths. In 3nuts one of the most robust backbone structures is combined with one of the most efficient lookup methods: random networks and search trees. These two structures complement each other excellently: Random networks are provably robust, but lack efficient lookup algorithms, and search trees efficiently support range queries, but are not robust. Another design goal was to forgo the use of heuristics wherever possible. So, 3nuts makes use of the simple and efficient load balancing provided by DHTs and uses a simple handshake operation [8] with provable guarantees for self-stabilization. Notably, these handshake operations allow to quickly recover from any degenerated state as long as the network is still weakly connected. 3nuts efficiently supports range queries and allows routing with small latency by adapting the overlay structure to the underlying physical network.

In Sec. II we describe three notions of locality relevant to p2p networks and discuss relevant literature. In Sec. III we describe 3nuts and its maintenance and in Sec. IV we underline the practicability by experimental results derived by a fully functional implementation, ready for practical use.

II. Locality in P2P Networks

Network locality: While the typical measure to evaluate routing algorithms is the hop number, this alone is not a good measure if the goal is to provide short response times. The reason for this is that a hop connecting peers in Italy and China has higher latency than a hop connecting peers in the same building. This leads to the definition of network locality.

Definition 1 *A p2p network provides network locality if the overlay structure is adapted to the physical network in order to reduce routing latencies.*

Network locality has been addressed early by the scientific community and several p2p networks providing network locality have been proposed. Pastry [5], based on the seminal work of Plaxton et al. [9], was among the first DHT based p2p networks providing network locality innately. Other networks have been extended to support network locality, e.g. [10] extends Chord [4] in this regard. The crux in providing network locality is to find latency wise close neighbors without generating too much additional network traffic.

Information locality: As we have seen DHTs are limited to exact match queries. So, data placement plays a key role when it comes to supporting complex queries, e.g. range queries.

Definition 2 *A p2p network provides information locality if closely related data is stored on network-wise close peers.*

The problem of supporting range queries in p2p networks has been identified early by several researchers [11]. Ratnasamy et al. proposed the trie based Prefix Hash Tree (PHT) [12], where prefixes of a trie are hashed onto an arbitrary DHT network. An advantage of their approach is that this way the load balancing functionality of DHTs can still be used. However, DHTs are inherently ill-suited for range queries and thus it is hardly surprising that the lookup in PHT is not as efficient as in DHTs, i.e. a lookup requires $\mathcal{O}(\log^2 n)$ hops, with n denoting the number of peers.

The skip list based Skip Graphs [13] belongs to the most prominent p2p networks supporting range queries efficiently. Yet, range query support in Skip Graphs is bought dearly by the loss of load balancing: Resources, e.g. data files, are managed by the peer hosting that resource respectively. Consequently, a peer hosting k resources has to maintain k nodes in Skip Graph. So, in a Skip Graph with n peers and m resources ($m \gg n$) a peer has to maintain $\mathcal{O}(k \log m)$ links, whereas the number of links to be maintained in DHT based networks typically is $\mathcal{O}(\log n)$ [4] or constant [6] per peer, and thus independent of the number of resources.

Several tree based p2p networks supporting range queries and load balancing at the same time have been proposed. We briefly discuss P-Grid [14] and DPTree [15]. P-Grid abstracts a binary trie structure defined by the data available in the network. Each peer is responsible for a particular prefix of the trie and maintains links to random peers of every subtree neighboring its own prefix. However, it is not clear if the links to subtrees selected in P-Grid are truly random and the load balancing mechanism used in P-Grid is based on heuristics. Without a central load balancing instance assigning prefixes to the peers, peers have to determine their prefix in a distributed manner. This can result in complex dependencies between all peers and the fact that P-Grid needs an extra bootstrapping mechanism for an initial network state underlines its complexity. DPTree [15] is inspired by balanced tree indexes. The tree structure is decoupled from the actual structure of the overlay by using a Skip Graph as overlay structure and choosing peer identifiers such that these represent paths from the root to leaves of the tree. Load balancing is done with a wavelet based mechanism to choose peer identifiers. Peers noticing to be overloaded may shed part of their load to neighboring peers. It may be criticized that the robustness under network dynamics is not verified and that the costs of rebuilding the tree upon structural changes remain unclear.

So, there exist numerous p2p networks that overcome the limitation of DHTs to exact match queries. Yet, the networks mentioned above either do not provide load balancing at all [13] or make use of complex heuristics [14], [15], which are in stark contrast to the simple and efficient load balancing provided by DHTs and often make a formal analysis impossible. The important thing to note here is that these networks — although they allow to process range queries efficiently — are not superior to DHTs in every respect.

Interest locality: In the Web certain data is intrinsically local, e.g. most of all greek web-sites are created in Greece and accessed from computers in Greece. Hence, it makes sense to store such data on peers located in Greece.

Definition 3 *A p2p network provides interest locality if peers can choose to provide lookup service and data storage for certain data. If peers choose to provide certain data, then the network allows efficient lookup to data relevant to a peer.*

Interest locality is rarely addressed in p2p networks. An exception is SkipNet [16] which is closely related to Skip Graphs [13]. So, SkipNet shares some shortcomings with Skip Graphs, i.e. the lack of load balancing. In fact the authors present a way to provide a constrained form of load balancing in SkipNet, but we will focus on interest locality. In SkipNet peers may choose arbitrary name id's. If all peers of a domain (e.g. '.ch') choose their name id to begin with their domain, then peers of the same domain will be neighbors in the id space. Using the domain as prefix for data as well allows to control data placement. Since the routing algorithm ensures that a query that has reached the target domain will never leave it again, SkipNet provides a form of interest locality. This, however, comes at the price of diminishing information locality: To retrieve all documents relevant to a query each domain has to be queried separately. So, there is a trade-off between information locality and interest locality in SkipNet.

Fig. 1 gives an overview of p2p networks and the supported types of locality. In the bottom line many networks supporting either network, information, or interest locality have been proposed. To the best of our knowledge 3nuts is the first network providing all three types of locality at the same time.

III. The 3nuts P2P Network

In 3nuts the peers build a distributed version of the prefix tree (trie) defined by the data available in the network. To make the distributed prefix tree robust, its nodes are replaced by random networks. The root of the tree is replaced by a random network containing all peers and forms a reliable backbone.

network	topology	network locality	information locality	interest locality
Gnutella	random graph	no	yes	no
Chord [4]	hypercube	no	no	no
Distance Halv.[6]	de Bruijn	no	no	no
Pastry [5]	mesh of trees	yes	no	no
Skip Graphs [13]	skip list/rings	no	yes	no
PHT [12]	DHT/trie	no	yes	no
DPTree [15]	Skip Graph	no	yes	no
P-Grid [14]	mesh of trees	no	yes	no
SkipNet [16]	skip list/rings	no	yes	yes
3nuts	tree / random	yes	yes	yes

Fig. 1. Overview of p2p networks and provided types of locality.

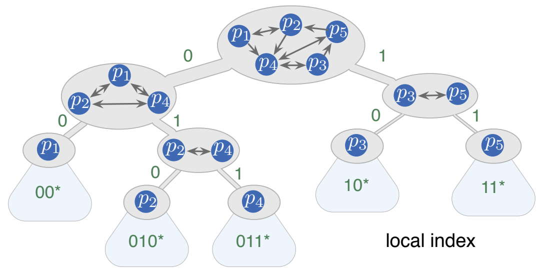

Fig. 2. Global view of a distributed prefix tree with 5 peers.

Then, peers are recursively assigned to subtrees using a DHT based load balancing mechanism until there is only a single peer left in every subtree (see Fig. 2 for a distributed prefix tree with 5 peers). So, each peer is assigned to a path from the root to a leaf of the distributed prefix tree. Every peer is responsible to manage the data with the prefix given by its path. 3nuts also allows to store data in internal nodes and each internal node of the tree has a particular peer which is responsible for managing data and creating new subtrees, etc.

Before we describe the assignment of peers to subtrees, the maintenance of random networks, and routing we introduce some notations. Let $p_1, \ldots, p_n$ be the set of peers and let T denote the distributed prefix tree with nodes $v_1, v_2, \ldots$. Note that each v_i represents a particular prefix and has a random network associated to it. By T_{v_i} we denote the subtree rooted at v_i and by $|T_{v_i}|$ the number of peers assigned to T_{v_i}. We define the load $w(T_{v_i}) \in \mathbb{N}$ of subtree T_{v_i} to be the number of data elements in T with prefix v_i. By "with high probability" (w.h.p.) we denote a probability $> 1 - n^{-c}$ for a constant c.

A. Peer Assignment, Load-balancing, and Responsibilities

The recursive assignment of peers to subtrees is done using distributed heterogeneous hash tables (DHHT) [17], an extended form of distributed hash tables (DHT) to support non-uniform weights. We give a brief description of DHTs and then describe the weighting extension, with focus on our application. Recall that we assign peers to subtrees (respectively data) while the usual approach is just the other way around. This allows to preserve a given ordering of data and thus overcome the limitation of DHTs to exact match queries.

We exemplify the peer assignment in an arbitrary node v of the distributed prefix tree with child nodes $v_1, \ldots, v_k$. Let

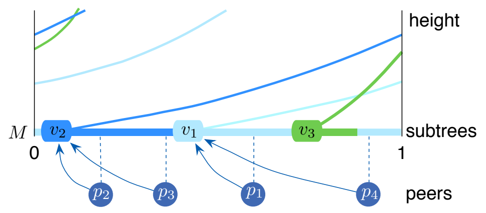

Fig. 3. Assigning peers to subtrees v_1, v_2, and v_3 using a DHHT. A smaller slope of functions means higher weight, i.e. $w(T_{v_1}) > w(T_{v_2}) > w(T_{v_3})$.

$p_1, \ldots, p_m$ be the peers that have been assigned to v. DHTs use a "two-sided" hashing into a continuous range $M = [0, 1)$ to assign peers to the subtrees rooted at $v_1, \ldots, v_k$. Peers and subtrees are mapped randomly into M by hash functions h_1 respectively h_2. Then, peers are assigned to the subtree which is closest to them in descending direction in M. So far all peers and subtrees are handled as if they are uniform. While this is reasonable for peers, it is likely that some subtrees hold more data than others and thus generate a higher load to the peers assigned to these subtrees. So, when assigning peers uniformly to subtrees, the load is not spread evenly among the peers. To take different weights of subtrees into account and make the number of peers assigned to a subtree T_{v_j} reflect its weight $w(T_{v_j})$, the scheme is extended as follows: Let $p'_i = h_1(p_i)$ and $v'_j = h_2(v_j)$ denote the position of peer p_i, respectively node v_j, in M. Then, we define a scaled distance function

$$L_w(p_i, v_j) = \frac{-\ln\left(\left(1 - (p'_i - v'_j)\right) \mod 1\right)}{w(T_{v_j})},$$

with $x \mod 1 := x - \lfloor x \rfloor$. Now peer p_i is assigned to the subtree rooted at the node v_j minimizing the term $L_w(p_i, v_j)$ (see Fig. 3). For peer p_i and node v_j we also refer to the value of this function as *height*.

If we extend this scheme to use double hashing, peer p_i is mapped into M using $p'_i = h_1(p_i)$ as before, but each subtree rooted at v_j, $1 \le j \le k$, has an individual hash function h_{v_j}. A peer then calculates its heights for each v_j at position $h_{v_j}(p'_i)$ and is assigned to the subtree minimizing the height. Using DHHTs with double hashing the following theorem is a direct consequence of Theorem 10 in [17].

Theorem 1 *Assigning peers to subtrees using the DHHT scheme in combination with double hashing it holds w.h.p. that*

$$\Pr[p_i \text{ is assigned to } v_j] = \frac{w(T_{v_j})}{\sum_{l=1}^{k} w(T_{v_l})} .$$

Hence, peers are assigned to subtrees with probabilities proportional to the weights of the subtrees. The runtime of the assignment using double hashing is linear in k, i.e. the number of subtrees. However, in our scenario k is a small constant.

Every node of the distributed prefix tree has a designated *responsible peer* which has to manage references to data stored in the node, create new subtrees when peers or data are inserted, and delete empty subtrees. The responsible peer for a node v is the peer that has been assigned to v with the lowest height. This choice is reasonable since this peer will be the last peer to leave the subtree T_v if $w(T_v)$ decreases or the load of subtrees rooted at v's siblings increases. Furthermore, the selected peers are chosen truly random and thus the responsibility for internal nodes is spread evenly among peers.

When assigning peers to subtrees, it is possible that no peer is assigned to a subtree T_v and thus no peer is responsible to manage the data in T_v. This will principally happen if $w(T_v)$ is small or T_v is a leaf of the tree. If there is such a vacant subtree T_v, then a peer is selected to be responsible for T_v by a mechanism called *shanghaiing*[1]: The peer that has the lowest height for T_v is selected to be shanghaied. In the scenario of Fig. 3 p_4 would get shanghaied to be responsible for subtree v_3. This choice is reasonable since p_4 will be the first to be assigned regularly to T_{v_3} if $w(T_{v_3})$ increases. A shanghaied peer is responsible for T_v until another peer is assigned regularly to T_v or a peer with lower height is shanghaied.

B. Maintaining Random Networks

All peers that have been assigned to a subtree are connected by a random network, i.e. each node v_i of the distributed prefix tree is represented by a random network. Here, we use d-out-regular multi-digraphs and maintain these using the Pointer-Push&Pull ($\mathcal{PP}$) operation [8]. $\mathcal{PP}$ is a simple handshake operation that is initiated by each peer periodically and transforms the network as described in Alg. 1. Note that a

Algorithm 1 Pointer-Push&Pull (peer p_1)

```
1: $p_2 \leftarrow$ random peer neighboring $p_1$
2: $p_3 \leftarrow$ random peer neighboring $p_2$
3: replace $p_2$ with $p_3$ in $p_1$'s list of neighbors
4: replace $p_3$ with $p_1$ in $p_2$'s list of neighbors
```

$\mathcal{PP}$ operation involves only two messages between two peers, carrying the information of one edge only. Thus $\mathcal{PP}$ can be used to replace the mandatory heartbeat (ping) messages used to verify the availability of neighbors in dynamic networks. Consequently, $\mathcal{PP}$ operations do not induce additional traffic to the network. In [8] it is shown that $\mathcal{PP}$ operations guarantee connectivity and generate (provably robust) truly random digraphs when applied repeatedly, as stated in Thm. 2.

Theorem 2 [8] *Let G_0^* be a d-out-regular connected edge labeled multi-digraph with n nodes. Then, applying random $\mathcal{PP}$ operations repeatedly will construct every graph of this domain with the same probability in the limit, i.e.*

$$\lim_{t\to\infty} P\left[G_0^* \xrightarrow{t} G^*\right] = \frac{1}{|\mathcal{MDG}^*_{n,d}|},$$

where t is the number of operations and $\mathcal{MDG}^*_{n,d}$ the set of all d-out-regular connected edge labeled multi-digraphs.

[1]Inspired by the english slang term, describing the common act of forcibly conscripting someone to serve a term working on a ship, usually after having been rendered senseless by alcohol or drugs, during the 19th century.

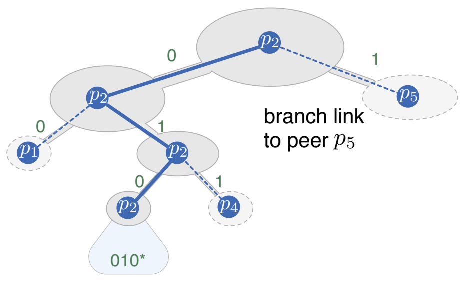

Fig. 4. The local view of peer p_2. Branch links are depicted by dashed lines.

An important consequence of Thm. 2 is that a peer will see every other peer participating in the same random network, i.e. subtree, over time. Hence, $\mathcal{PP}$ operations constitute an excellent tool to spread information about the tree structure, etc. among peers without inducing additional traffic to the network. The latter is the main reason for us to prefer multi-digraphs over the more common domain of regular graphs.

C. A Peer's Local View

Since a peer is assigned to a path in the distributed prefix tree it only has a local view of the network. We refer to the nodes a peer has been assigned to (regularly or shanghaied) as *trunk nodes*. For each trunk node a peer maintains a *trunk node table* with information about node id, weight, subtrees, and references to data in case of the responsible peer. Furthermore, the trunk node table contains the following lists:

a) Responsibility list: A list of the r peers with the highest responsibility, i.e. those that have been assigned to the node with lowest height in the parent node (see Sec. III-A).

b) Random neighbors: A list of d neighboring peers in the random network corresponding to this node of the tree. The list is used to perform $\mathcal{PP}$ operations and thus the entries are guaranteed to be truly random by Thm. 2.

c) Branch links (random and local): A peer maintains branch links to some peers of every subtree neighboring its own trunk nodes. Here, we distinguish *random* and *local* branch links. The former point to truly random peers of a subtree. A single list entry contains the id and weight of the subtree, and the address of the corresponding peer. Local branch links are similar to random branch links with the exception that these do point to latency wise close peers of a subtree. Fig. 4 shows the local view with trunk nodes and branch links of peer p_2 (cf. Fig. 2 for the global view).

To join the network, a peer contacts an arbitrary peer p of the network and proceeds as described in Alg. 2. The joining peer copies p's trunk node table for the root of the distributed prefix tree and then, based on the subtrees and weights given in this table, chooses a subtree using the DHHT scheme. Using the list of branch links it is ensured that a peer p' of this subtree can be contacted. Then, the same procedure is continued in the root node of the chosen subtree with peer p' and so on until the joining peer is the only one assigned to a subtree.

Algorithm 2 Join(peer p)

```
1: $v \leftarrow$ root of tree
2: initialize trunk node table for $v$ by copying $p$'s table
3: **while** number of peers in $v > 1$ **do**
4:     $v \leftarrow$ root of subtree determined using DHHT
5:     $p' \leftarrow$ peer participating in $v$
6:     contact $p'$ in $v$ and initialize trunk node table for $v$
7:         by copying table of $p'$
```

D. Self-Stabilization with Pointer-Push&Pull Operations

To cope with churn, failing peers, load changes, and changes in the tree structure it is crucial to have a simple and efficient protocol to maintain and stabilize the network. For this, 3nuts makes use of the excellent communication properties of random networks and the properties of the $\mathcal{PP}$ operation. Whenever two peers communicate during a $\mathcal{PP}$ operation their responsibility list, branch link list, and weights of the subtrees are piggy-backed to the messages and used to update their trunk node tables as described below. Notably, the following update procedure, based on local handshake operations only, guarantees to restabilize the network from any degenerate state as long as it is at least weakly connected.

A peer p_1 that has communicated with a peer p_2 in node v will update its trunk node table as follows. Let $T_{v'}$ be the subtree of T_v that p_2 has been assigned to. The random branch link corresponding to $T_{v'}$ is set to point to p_2. This way branch links are continuously replaced with peers that are ensured to be reachable. Entries for subtrees not existent in the branch link list of p_1 are inserted and entries that have been identified to be dead during $\mathcal{PP}$ operations or routing are replaced. The combination of update procedure and $\mathcal{PP}$ guarantees that random branch links point to truly random peers and that over time all peers participating in T_v are contacted. Moreover, latencies are measured during $\mathcal{PP}$ operations. When the latency from p_1 to p_2 is lower than the latency of the current local branch link to $T_{v'}$, then the link is replaced with p_2. From Thm. 2 we know that every peer participating in T_v will be met over time and thus for every subtree the latency wise closest peer will be identified.

Furthermore, weights of subtrees are updated and entries of the responsibility list are replaced when peers with higher responsibility are found in p_2's list or added if p_1's list has less than r entries. When changes to the weights or branch link list have been made, p_1 recalculates its assignment to the subtrees and changes its path, if necessary.

The update procedure described above guarantees the network to reach a stable state as long as all random networks representing nodes of the distributed prefix tree are connected. Although $\mathcal{PP}$ operations guarantee connectivity, random networks may be disconnected by peers failing simultaneously. First of all note that it is difficult to detect if a random network has been partitioned into two connected components G_1 and G_2 without global knowledge. Even if there were a simple and distributed way to detect this, rejoining G_1 and G_2 bears the danger of disconnecting other parts of the network since one edge of G_1 has to be changed to point to G_2. This however may disconnect G_1. Multi-digraphs offer a solution to this dilemma: self-loops, which may be removed without risking connectivity. So, random networks in 3nuts are rejoined 'proactively' as follows. Once again consider peers p_1 and p_2 that communicated during a $\mathcal{PP}$ operation in node v and let T'_v be the subtree of T_v that p_1 has been assigned to. If p_1 has a self-loop in the random network representing v' then with probability $\frac{1}{2}$ the self-loop is replaced with the branch link to $T_{v'}$ in p_2's trunk node table.

Using the maintenance protocol with the extension to rejoin random networks, 3nuts is able to recover from any degenerate state as long as the random network representing the root node is connected. Unfortunately, we are not able to give a formal proof for the number of $\mathcal{PP}$ operations necessary to re-stabilize a network. The exchange of information using $\mathcal{PP}$ operations is closely related to randomized rumor spreading [18]. A major difference making a formal analysis difficult is that in our case the underlying network is not a complete graph and changes over time. Yet, if we assume the network to be truly random we expect the dissemination of information by $\mathcal{PP}$ to behave comparably as in [18], where $\mathcal{O}(n \ln \ln n)$ messages are needed to spread a rumor among n nodes w.h.p.

E. Routing

The lookup algorithm is given by Alg. 3. It is started at an arbitrary peer p and the only parameter is the identifier key of a data element, which describes a path in the distributed prefix tree T. To reach the node v storing key, p follows the path key in its local view of T until a leaf node is reached. This leaf node can be a branch link or a trunk node. In the former case the lookup is forwarded to the corresponding branch link. In the latter case the lookup is forwarded to the peer responsible for the trunk node (since most data resides in leaves, it is likely that the responsible peer is reached directly).

Algorithm 3 Lookup(key) at peer p

```
1: **if** $p$ has branch link to a peer $p'$ sharing longer prefix with $key$
   **then** forward Lookup( $key$ ) to $p'$
2: **else**
3:     $v \leftarrow$ last node of path $key$ in local view of $p$
4:     $p' \leftarrow$ peer responsible for $v$
5:     **if** $p = p'$ **then return** $p$
6:     **else** forward Lookup( $key$ ) to $p'$
```

Theorem 3 *In a 3nuts network with n peers the number of hops for a lookup operation is bounded by $\mathcal{O}(\log n)$ w.h.p.*

Proof: We bound the number of hops needed to reach a subtree containing at most $\frac{n}{2}$ peers. Let $P = (v_1, \ldots, v_i, v_j, \ldots, v_k)$ be the path starting at the root of T leading to the target node v_k. We choose v_i and v_j such that $|T_{v_i}| \geq \frac{n}{2}$ and $|T_{v_j}| \leq \frac{n}{2}$, i.e. T_{v_i} is the smallest subtree rooted on P containing at least $\frac{n}{2}$ peers.

The lookup starts at an arbitrary peer p in v_1. Let p' be the peer reached by the first hop. Since $|T_{v_i}| \geq \frac{n}{2}$ and branch links point to truly random peers, p' will lie in T_{v_i} with probability of at least $\frac{1}{2}$. If p' does not lie in T_{v_i}, the same argumentation holds for the next hop from p'. So, we reach T_{v_i} with one hop with probability $\geq \frac{1}{2}$, with two hops with probability $\geq 2^{-2}$, and with k hops with probability $\geq 2^{-k}$. Thus, we have

$$E\left[\#\text{hops to reach peer in } T_{v_i}\right] \leq \sum_{k=1}^{k=\frac{n}{2}-1} k2^{-k} \leq 2 \ .$$

Since T_{v_i} is the *smallest* subtree with $|T_{v_i}| \geq \frac{n}{2}$, once we reached a node in T_{v_i} one more hop is sufficient to reach T_{v_j} with $|T_{v_j}| \leq \frac{n}{2}$. So, in expectation at most 3 hops are needed to halve the number of peers. Due to the recursive structure of 3nuts the same line of arguments holds for subtree T_{v_j}. This implies that after $\log n$ iterations respectively an expected number of $3\log n$ hops, the lookup has reached v_k.

It remains to show that $\mathcal{O}(\log n)$ hops are sufficient to reach v_k with high probability. We have seen that

$$\Pr[\#\text{peers is halved within three hops}] \geq \frac{1}{2} + \frac{1}{4} = \frac{3}{4} \ .$$

Dividing the lookup into sequences of three hops allows us to reduce the analysis to a sequence of mutually independent random variables $X_1, X_2, \ldots, X_{c\log n}$ taking values 0 and 1 with $\Pr[X_i = 1] = \frac{3}{4}$ and $X = \sum_{i=1}^{c\log n} X_i$. The expected number of successful steps is given by $E[X] = \frac{3}{4}c\log n$. Choosing $\delta = 1 - \frac{4}{3c}$ and applying Chernoff bounds we have

$$\begin{aligned}\Pr[X \leq \log n] = \Pr[X \leq (1-\delta)E[X]] &\leq e^{-\frac{1}{2}\left(1-\frac{4}{3c}\right)^2 E[X]} \\ &\leq n^{-\frac{121}{600}c} \leq n^{-c'}.\end{aligned}$$

So, the probability to be successful less than $\log n$ times is polynomially small in n. This implies that the lookup needs at most $3c\log n$ hops w.h.p. if we choose $c \geq 5$. ■

It is important that the bound given by Thm. 3 holds regardless of the structure of the distributed prefix tree T since real world data will not be uniformly distributed. The reason that Thm. 3 holds for skewed data distributions is that branch links point to random peers of subtrees. Due to the properties of the $\mathcal{PP}$ operation (see Thm. 2) and the way branch links are maintained (see Sec. III-C), we can guarantee these to be truly random. Actually branch links are not only random but continually change (recall that whenever peers p and p' communicate during a $\mathcal{PP}$ operation p will set the branch link corresponding to the subtree p' has been assigned to, to point to p'). This feature implies that routing paths are continually changing when random branch links are used for routing and thus the routing load will be spread evenly among peers. A typical measure with respect to routing load is the congestion.

Definition 4 *The congestion of a peer is the probability that it is active in the routing of a random lookup operation started at a random peer. The congestion of the network is the maximum congestion over all its peers.*

Assuming that the network is in a stable state and branch links are truly random the following theorem holds.

Theorem 4 *The congestion of 3nuts is bounded by $\mathcal{O}(\frac{\log n}{n})$.*

Proof: From Thm. 3 we know that the number of hops is bounded by $k \leq c\log n$. Let $T_{v_1}, \ldots, T_{v_k}$ be the subtrees reached during these k hops. Now consider an arbitrary peer p_j. Note that p_j can only get active once during a lookup. To become active during the i-th hop p_j must have been assigned to T_{v_i} and must have been chosen as branch link by the peer reached by the previous hop. The probability that p_j has been assigned to T_{v_i} is $|T_{v_i}|/n$. If the network is in a stable state, the probability for a peer assigned to T_{v_i} to become active during the i-th hop is $1/|T_{v_i}|$. Thus we have

$$\Pr[p_j \text{ is active during hop } i] = \frac{|T_{v_i}|}{n}\frac{1}{|T_{v_i}|} = \frac{1}{n}.$$

Consequently, the probability for p_j to become active during the whole lookup is given by $\frac{k}{n} = \frac{c\log n}{n}$. ■

Alg. 3 can be easily extended to perform range queries. To search for all data in a range $[x, y]$ the longest common prefix z of x and y is calculated. Then, the query is routed to the node v of the prefix tree representing z. Note that T_v is the smallest subtree containing all data in the range $[x, y]$. Starting from v, the query is forwarded to all subtrees holding data in the range $[x, y]$ in parallel until the leaf nodes of T_v are reached. Peers receiving the lookup message send their list of data to the peer that originated the lookup respectively forward the lookup to the responsible peer in case of internal nodes of T_v.

F. Locality in 3nuts

Since the peers build a distributed prefix tree closely related data elements are stored on network-wise close peers.

Theorem 5 *Let d be the distance of two data elements x and y in the tree metric. Then, x and y can be reached within d hops from one another.*

Proof: Let v be the node of T representing the longest common prefix of x and y. Note that v is present in the local view of the peer p that is responsible for x since v is a prefix of x. So, no hops are needed to reach node v from p. From node v at most d hops are needed to reach y since each hop will advance at least one level in T and the distance between v and y is bounded by d. Recall that in any case the maximum hop number is bounded by $\mathcal{O}(\log n)$ w.h.p. ■

Recalling the types of locality introduced in Sec. II, Thm. 5 implies that 3nuts provides information locality and we have already seen that range queries can be processed efficiently.

3nuts provides network locality through the list of local branch links in the trunk node tables, which point to latency wise close peers of neighboring subtrees and may be used for routing instead of random branch links. Initially, a local branch link list is a copy of the random branch link. The quality of local branch links is then improved via $\mathcal{PP}$ operations as described in Sec. III-C. Most importantly the update procedure does not induce additional traffic but guarantees to find the closest peer for every subtree. In Sec. IV we compare latencies when routing with random and local branch links.

3nuts provides interest locality by allowing peers to *volunteer* for the responsibility of nodes of the distributed prefix tree T. Volunteering does not relieve a peer from participating in the regular assignment using DHHTs and thus induces additional workload to a peer. Yet, a peer p volunteering for the responsibility of a node v will drastically decrease its access times to parts of T that are close to v. Peer p will actively participate in the path starting at the root of T leading to v. For nodes of this path that are not coincident with the path p has been assigned to, p has to maintain additional trunk nodes.

IV. Experimental Evaluation

To verify the practicability of 3nuts we used a prototype implementation in Java, ready for practical use and available at http://3nuts.upb.de. For the experiments the degree of random networks was set to $d = 3$ and if not stated otherwise, the network consisted of $n = 2^{14}$ peers. Each peer stored five data elements giving a total of $81,920$ data elements. Measurements have been performed multiple times and curves represent the mean. If no error bars are given the variance is negligible. Most measurements have been performed with several types of data to verify the impacts of data distribution and degree of the distributed prefix tree. These are:

Binary tree (uniform): A binary tree with data elements representing binary strings of length 40, chosen u.a.r.

Binary tree (Zipf): A binary tree of data elements representing binary strings of length 52, chosen according to a Zipf distribution as follows. The leaf nodes of a complete binary tree of depth 20 have been assigned probabilities following a Zipf distribution with exponent 1. Data elements are placed by choosing a prefix of length 20 according to these probabilities and concatenating a binary random string of length 32.

Dictionary: A tree generated by choosing data uniformly at random from a list of English words (we used the word list of ispell). The dictionary tree has degree up to 26.

Routing Fig. 5 shows the average number of hops needed by the lookup for different data distributions and networks up to 2^{14} peers. A single measurement for a fixed network size and data distribution was done by performing 10^6 random lookups chosen from all possible combinations of peers and data. The curves representing the binary trees are almost equal, implying that the DHHT based load balancing performs excellently and the scalability is not affected by non-uniform data distributions. In case of the dictionary tree fewer hops are needed since the tree has substantially higher degree and thus lower depth than the binary trees.

To evaluate the benefit of using local instead of random branch links we used GT-ITM [19] to model the underlying physical network. Fig. 6 shows the average latencies for the three tree types when using random respectively local branch links. The average latency measured with random links in binary trees exceeds the one measured for the dictionary tree by a factor of 1.8. This is explained by the larger number of hops needed in the comparatively deep binary trees. When using local branch links, latencies are reduced significantly for all tree types. Notably, average latencies measured for dictionary and binary trees then only differ by a factor of 1.1, i.e. the binary trees benefit more from the use of local links. This is explained by the fact that the last few hops are by far the most "expensive" ones since the number of peers to choose from decreases with each hop, i.e. it is more unlikely that latency wise close peers can be found.

Load Balancing Fig. 7 shows the distribution of data load. Since the DHHT based load balancing makes use of pseudo-random hash functions there are some peers exceeding the average load of 5. Anyhow, 90% of the peers have at most twice the average load. While having peers exceeding the average load is not optimal, one has to recall the simplicity of the DHHT scheme: The decisions a peer makes when choosing its path are completely independent of the decisions made by other peers. So, a peer usually does not have to change its path when further peers join. Coping with not 100% fair load balancing is the price to pay for this simplicity. Yet, simplicity can be crucial to keep the network stable under churn.

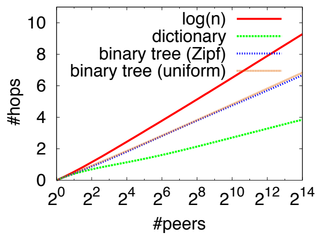

Fig. 5. Average number of hops needed by the lookup operation for different sized networks.

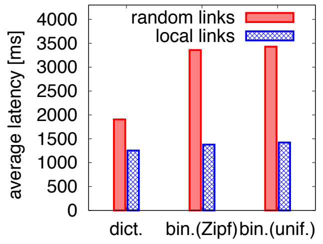

Fig. 6. Average latency of lookups: random branch links vs. local branch links.

Degree A peer's degree, i.e. the number of neighbors, depends on the degree and depth of the distributed prefix tree: high degree involves a large number of branch links and high depth involves a larger number of trunk nodes. Fig. 8 shows the sum of branch links and neighbors in the random networks per peer. For the uniform binary tree the average degree is 42 (12 branch links plus 30 random links in trunk nodes). The Zipf distributed tree leads to slightly increased degree. This is explained by the higher depth of the resulting distributed prefix tree. In case of the dictionary tree the high fan out imposes a large number of branch links. Surely, a peer's degree in a standard DHT network [4] is lower. Yet, considering the additional features of 3nuts, e.g. information and network locality, self-stabilization by local handshake operations, the "costs" for these features, i.e. the increased degree, are negligible. Moreover, links in 3nuts are very easy to maintain: While links in most p2p networks have to point to one particular peer, a link in 3nuts may always point to a random peer out of a large set of candidates.

Self-Stabilization To verify robustness and self-stabilization a network of 10^4 peers was generated and 25% of the peers were removed simultaneously. Fig. 9 shows the evolution of data availability which was checked by performing 10^6 random lookups. A lookup was considered as failed whenever an invalid branch link was encountered. Trunk node tables

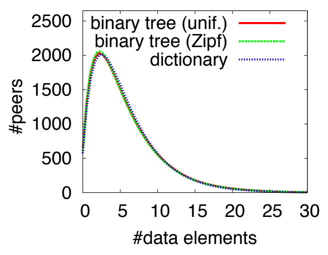

Fig. 7. Load balancing: number of data elements per peer (average load is 5).

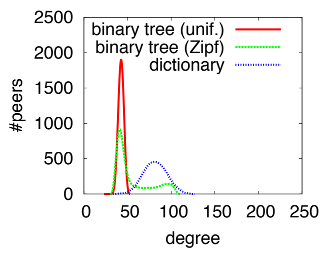

Fig. 8. Degree distribution: number of branch links and random neighbors per peer.

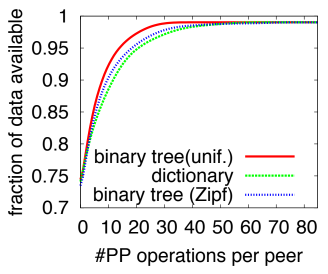

Fig. 9. Evolution of data availability after failure of 25% of peers.

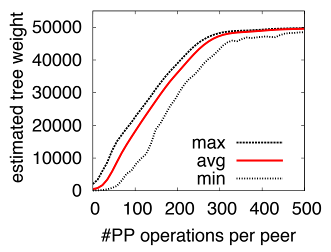

Fig. 10. Evolution of estimated tree weight when inserting 10^4 peers.

were only repaired by $\mathcal{PP}$ operations. The different tree types behave almost equal. Right after the removal of peers about 75% of the data remaining in the network is still available and after 30 to 50 $\mathcal{PP}$ operations per peer, data availability of 99% is reached. In a final experiment we generated a network of 10^4 peers and 50,000 data elements. All peers joined simultaneously, so this experiment can be considered as an extreme example of churn. Note that at the beginning of this experiment the distributed prefix tree is just the root node and peers are only connected by the random network representing the root. Fig.10 shows the evolution of the weight of the tree estimated by the peers. About 300 $\mathcal{PP}$ operations per peer are sufficient for the peers to find their position in the tree and get a coherent view of the distributed prefix tree.

V. Concluding Remarks

3nuts combines random networks, prefix trees, and DHTs, to overcome their individual shortcomings. To the best of our knowledge 3nuts is the first p2p network providing interest, network, and information locality at the same time. The practicability of 3nuts has been affirmed by a prototypical implementation ready for practical use, experimental evaluation, and verification on a mathematical level where possible. 3nuts has been designed around the $\mathcal{PP}$ operation, whose properties make it an excellent maintenance operation for dynamic networks. Replacing the heartbeat (ping) messages between peers, $\mathcal{PP}$ operations are used to:

- maintain truly random networks to replace nodes of the data tree and thus make the network robust,
- exchange information among peers, give peers a coherent view of the tree structure and stabilize the network,
- maintain branch links and guarantee them to be truly random, thus allow efficient routing,
- and measure round trip times (RTT) to adapt the overlay to the underlying physical network.

A possible drawback is the potentially high degree, which can be caused by highly skewed data distributions resulting. Yet, maintenance of links is comparably cheap in 3nuts and higher degree implies higher robustness. If necessary, the degree can be reduced by using radix or balanced trees.

References

[1] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *IPTPS '03: Proc. of the 2nd Int. Workshop on p2p Systems*, 2003.

[2] P. Mahlmann and C. Schindelhauer, "Peer-to-peer networks based on random transformations of connected regular undirected graphs," in *SPAA '05: Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architectures*, 2005, pp. 155–164.

[3] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS '02: Proc. of the 16th Int. Conference on Supercomputing*. ACM Press, 2002, pp. 84–95.

[4] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of SIGCOMM'01*, ser. Computer Communication Review, R. Guerin, Ed., vol. 31, 4. ACM Press, Aug. 27–31 2001, pp. 149–160.

[5] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329–350.

[6] M. Naor and U. Wieder, "Novel architectures for p2p applications: The continuous-discrete approach," *ACM Trans. Algorithms*, vol. 3, no. 3, p. 34, 2007.

[7] S. Schmid and R. Wattenhofer, "Structuring unstructured peer-to-peer networks," in *HiPC'07: Proceedings of the 14th international conference on High performance computing*. Springer, 2007, pp. 432–442.

[8] P. Mahlmann and C. Schindelhauer, "Distributed random digraph transformations for peer-to-peer networks," in *SPAA '06: Proc. of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, 2006.

[9] C. G. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA'97: Proc. of the 9th ACM Symp. on Parallel Algorithms and Architectures*, 1997.

[10] A. Montresor, M. Jelasity, and O. Babaoglu, "Chord on demand," in *P2P '05: Proc. of the 5th IEEE Int. Conf. on Peer-to-Peer Computing*, 2005, pp. 87–94.

[11] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, "Complex queries in dht-based peer-to-peer networks," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 242–259.

[12] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker, "Prefix hash tree," in *PODC '04: Proc. of the 23rd ACM Symp. on Principles of Distributed Computing*. ACM, 2004, pp. 368–368.

[13] J. Aspnes and G. Shah, "Skip graphs," *ACM Transactions on Algorithms*, vol. 3, no. 4, p. 37, 2007.

[14] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-Grid: a self-organizing structured p2p system." *SIGMOD Record*, vol. 32, no. 3, pp. 29–33, 2003.

[15] M. Li, W.-c. Lee, and A. Sivasubramaniam, "DPTree: A balanced tree based indexing framework for peer-to-peer systems," in *ICNP '06: Proc. of the Int. Conf. on Network Protocols*. IEEE, 2006, pp. 12–21.

[16] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties," in *USENIX Symposium on Internet Technologies and Systems*, 2003.

[17] C. Schindelhauer and G. Schomaker, "Weighted distributed hash tables," in *SPAA '05: Proceedings of the seventeenth annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2005, pp. 218–227.

[18] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking, "Randomized rumor spreading," in *FOCS '00: Proc. of the 41st Symp. on Foundations of Computer Science*, 2000, p. 565.

[19] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *INFOCOM '96: Proc. of 15th Joint Conf. of the IEEE Computer Societies*, vol. 2, 1996, pp. 594–602.