Project Number 001907

DELIS
Dynamically Evolving, Large-scale Information Systems

Integrated Project
Member of the FET Proactive Initiative Complex Systems

Deliverable D6.2.2

Final Report on
Concepts for Advanced Overlay Networks that Combine Locality with Hashing and Randomization
Start date of the project: January 2004
Duration: 48 months
Project Coordinator: Prof. Dr. math. Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable: December 2005
Actual submission date: January 2006
Dissemination level: PU – public

Work Package 6.2: Enhanced Distributed Hash Tables for Keyword Search
Participants: University of Paderborn (UPB), Germany
Max Planck Institute for Computer Science (MPII), Saarbrücken, Germany
Computer Science Institute (CTI), Patras, Greece
Authors of deliverable: Peter Mahlmann (mahlmann@upb.de)
Christian Schindelhauer (schindel@upb.de)
1 Introduction and Overview

In this final report we present our new approaches for maintaining random graphs as network structure of peer-to-peer networks, improving distributed hash tables (DHT) to support heterogeneity, and supporting locality in peer-to-peer networks.

We start by motivating the use of random graphs in peer-to-peer networks. Having pointed out the benefits of random graphs, we present the Flipper operations, which are able to maintain random graphs using a simple local update operation involving four peers only. Furthermore, we show that the Flipper operations will turn any graph into an expander graph when applied repeatedly.

We then give a brief introduction to distributed hash tables (DHTs) and present the state of the art. Then, we present our new approach, called distributed heterogeneous hash tables (DHHTs) which extends the original scheme of DHTs with weighting such that the amount of data assigned to a server is proportional to its computing, storage or network capabilities. The load balancing achieved by the DHHT scheme is provably fair.

Finally, we define three aspects of locality in the area of peer-to-peer networks: network locality, information locality, and interest locality. After briefly analyzing existing peer-to-peer networks concerning their capabilities to support complex queries and these types of locality, we give a brief description of the 3nuts peer-to-peer network which supports complex queries as well as locality. Therefore, 3nuts makes extensive use of the two techniques introduced before — the Flipper operations for maintaining random graphs and DHHTs for load balancing.

2 Random Graphs in Peer-to-Peer Networks

In this section we analyze random graphs with respect to the use as link structure in a dynamic peer-to-peer networks. Then, we present our new results for distributed construction and maintenance of random graphs as a link structure.

2.1 Identifying Random Graphs as a Convenient Network Structure for Peer-to-Peer networks

Random Graphs always played an important role in peer-to-peer networks. For example the Gnutella network [Gnu] used a random graph as network structure. Today, random graphs can be found in the JXTA peer-to-peer networking suite of SUN corporation [JXT], for example. There, random graphs are used as a reliable backbone network, on which users of the JXTA suite can implement their own peer-to-peer applications.

Random graphs have been studied extensively in computer science [Bol05] and have many properties which are in particular useful for distributed networks. There are several classes of random graphs (e.g. directed/undirected graphs, simple/multi graphs, regular graphs, etc.). Here, we restrict ourselves to the domain of undirected regular random graphs. We will now give a brief overview of some of these desirable properties and state why they are advantageous.

- Simplicity:
  A random graph is an extremely simple graph structure. Surely, there are other simple structures like a stars or rings, however these structures have several shortcomings when they are used for peer-to-peer networks. This simplicity is especially useful in dynamic networks like peer-to-peer networks, where peers, e.g. nodes, are joining and leaving the network all the time. This is because every time a peer disconnects from the network, its former neighbors need to repair their neighborhood in order to maintain the desired network topology. This repair process is in general non trivial, since the peer which has to repair its neighborhood has no global knowledge of the network. Therefore, the peer has to search the network for...
the correct replacement of its former neighbor. Things are different in a random graph. In a random graph there is no predetermined neighborhood for a peer. This implies that a peer that is missing one of its neighbors can choose an almost arbitrary other peer as replacement. Note, that in fact not every peer may be suitable, e.g. peers which are already neighbored may not be chosen if one restricts to simple graphs, i.e. one does not allow the multiple occurrence of edges (what leads to multi-graphs).

- **Small diameter:**
  An undirected random regular graph has a small diameter with high probability. More precisely, if the degree $d$ of the graph is bounded by $d \in \omega(1)$ the diameter is logarithmic in the number $n$ of nodes with $p \geq 1 - o(1)$ and if $d \in \Omega(\log n)$ the diameter is logarithmic with probability $p \geq 1 - n^{-o(1)}$ (see [Wor99]). The usefulness of a logarithmic diameter is apparent, since it implies that any node of the graph can reach every other node of the graph quickly.

- **Expansion:**
  Graph expansion is an even more powerful property than a small diameter. First of all, let us define the notion of expander graphs. Therefore, let $G = (V, E)$ be a a graph with node set $V$, $|V| = n$ and edge set $E \subseteq (V \times V)$.

**Definition 1**

1. For $S, T \subset V$ denote the set of all edges between $S$ and $T$ by $E(S, T) = \{ \{u, v\} | u \in S, v \in T, \{u, v\} \in E\}$.
2. The edge boundary of a set $S \subset V$, denoted by $\partial S$, is $\partial S = E(S, \bar{S})$ with $\bar{S} = V \setminus S$.
3. A graph $G = (V, E)$ is an expander graph or provides the expansion $\beta > 0$ if for all node sets $S$ with $|S| \leq |V| / 2$

$$|\partial S| \geq \beta |S|.$$ 

Informally, in an expander graph every subset of nodes has a relatively large neighborhood in the rest of the graph. The expansion property is extremely useful and has found many applications in computer science. For the use in peer-to-peer networks the three most important properties of expander graphs are: logarithmic diameter, high vertex connectivity and small mixing time of random walks. These properties imply a high robustness against the adversarial deletion of nodes, i.e. if nodes are removed from the graph, the resulting graph will remain connected with high probability.

2.2 **State of the Art: Random Graph Transformations for Peer-to-Peer Networks**

Before we discuss the state of the art let us point out some crucial properties for a graph transformation aiming at building random graphs in a peer-to-peer network.

- **Soundness:**
  No transformation maps to graphs which are not in the domain space. For $d$-regular undirected connected graphs, this means that each operation preserves degree $d$ at every node and there is not even the slightest (small probability) chance to disconnect parts from the graph.

- **Generality:**
  The random transformation process does not converge to a specific graph. All graphs are reachable and in the limit all graphs occur with some non-zero probability. This requirement can be tightened to uniform generality where in the limit all graphs occur with the same probability.
• **Feasibility:**
The random transformation process can be described by a simple (distributed) routine changing only a small number of edges of the graph. Its implementation in a distributed network should be straightforward.

• **Convergence rate:**
Only a polynomial number of transformations is necessary to achieve an approximation of the ultimate distribution of all graphs.

As noted in section 2.1 random graphs are used in the SUNs JXTA suite, for example. Surprisingly, the process used in JXTA, which is widely used, has never been analyzed with respect to these features. Identifying such transformations meeting all these requirements gives an approximate solution to the problem of computing a random probability distribution over all graphs of a kind, as being solved by Steger et al. [SW99] for the problem of generating all \(d\)-regular random graphs.

To provide a random transformation for \(d\)-regular undirected graphs without connectivity consider the following transformation introduced by McKay [McK81] and used by [GMS04] page 215 (there called “rewiring”).

**Definition 2 (Simple Switching)** Choose two random edges \(\{u, v\}, \{u', v'\}\) of the graph. If \(\{u, v'\}\) and \(\{u', v\}\) do not exist in the graph then erase \(\{u, v\}, \{u', v'\}\) and insert edges \(\{u, v'\}\) and \(\{u', v\}\).

In [MW90] this Simple Switching is used to generate random \(d\)-regular graphs with \(d \in O(n^{1/3})\) and it takes an expected time of \(O(nd^3)\) per graph to generate a uniform distribution over all \(d\)-regular graphs. Simple Switching preserves the degree of each node and does not preserve connectivity. Its convergence speed is polynomial in the number of nodes which follows from the results of [CDG05, MW90].

This Simple Switching is feasible if the graph is given as a data structure on a single machine. However, when the graph constitutes an interconnection network of computers, this procedure is not feasible anymore. As long as all nodes are connected one can choose two random edges by performing a random walk (with an appropriate length which is at least the mixing time of the graph). But during these operations Simple Switching may disconnect parts from the network. Then, without extra network connections the network cannot be rejoined anymore. Our point is that feasibility in terms of distributed algorithms implies maintaining connectivity at all stakes.

Another approach to build low diameter random graphs in peer-to-peer networks is given by Pandurangan, Raghavan, and Upfal in [PRU01]. They present a simple scheme for peers to build a low degree peer-to-peer network in a distributed fashion, and prove that it results in connected networks of constant degree and logarithmic diameter with high probability. However, they can not guarantee connectivity and a so called host server is used in their scheme. This host server — while not having global knowledge of the network topology — is a central component and violates the peer-to-peer paradigm.

### 2.3 New results

In [DELIS-TR-0148] we introduce the Flipper operations — a simple random graph transformation, which is able to build and maintain \(d\)-regular connected undirected random graphs as a distributed network structure. The Flipper operation is able to overcome the shortcomings of the approaches mentioned in Section 2.2 and provides soundness, generality, feasibility, and fast convergence.

#### 2.3.1 Uniform Generation of Regular Connected Graphs

Let \(G = (V, E)\) be a \(d\)-regular connected undirected graph and let \(N(v), v \in V\) denote the set of nodes neighboring \(v\) in \(G\). Then, the following algorithm describes the Random 1-Flipper operation.
Algorithm 1 Random 1-Flipper

Choose random edge \( \{u_2, u_3\} \in E \)
Choose random node \( u_1 \in N(u_2) \setminus \{u_3\} \)
Choose random node \( u_4 \in N(u_3) \setminus \{u_2\} \)

if \( \{u_1, u_3\}, \{u_2, u_4\} \notin E \) then

\[
E \leftarrow E \setminus \{\{u_1, u_2\}, \{u_3, u_4\}\}
\]

\[
E \leftarrow E \cup \{\{u_1, u_3\}, \{u_2, u_4\}\}
\]

Informally, the Random 1-Flipper operation chooses a random path \( P \) of three edges in \( G \) and then flips the two outer edges of \( P \) (see Figure 1). By definition this operation is sound for the domain of \( d \)-regular connected undirected graphs, i.e. preserves connectivity and regularity.

In [DELIS-TR-0148] we show, that the Random 1-Flipper operation is able generate every \( d \)-regular connected undirected Graph. Furthermore, we have proven the following theorem in [DELIS-TR-0148] by analyzing the Markov Process described by the Random 1-Flipper operation, yielding an even stronger result. Let \( G_0 \rightarrow G \) denote the predicate that \( G \) is derived from \( G_0 \) by applying \( i \) Random 1-Flipper operations and let \( C_{n,d} \) denote the set of all connected \( d \)-regular graphs with \( n \) nodes.

**Theorem 1** Let \( G_0 \) be a \( d \)-regular connected graph with \( n \) nodes and \( d > 2 \). Then in the limit the Random 1-Flipper operation constructs all connected \( d \)-regular labeled graphs with the same probability, i.e.

\[
\lim_{t \rightarrow \infty} P[G_0 \xrightarrow{i} G] = \frac{1}{|C_{n,d}|}.
\]

Theorem 1 shows that the Random 1-Flipper operation provides uniform generality for the domain of \( d \)-regular connected undirected graphs, i.e. it will generate every such graph with the same probability when applied repeatedly. This result in turn implies, that the 1-Flipper operation generates expander graphs in the limit and therefore a Flipper maintained random network inherits all the advantageous properties of expander graphs. Last but not least, the 1-Flipper is the first decentralized algorithm for the uniform generation of connected regular graphs.

Due to its simplicity the implementation in a peer-to-peer network is straightforward. However, up to now the convergence speed of the Random 1-Flipper operation is unknown. This open question motivated the generalization of the 1-Flipper described in the next section.

**2.3.2 Fast Construction of Expander Graphs**

In this section we present a generalization of the 1-Flipper operation for which we can show a polynomial bound on the convergence speed towards an expander graph. For this, we extend the random path of the 1-Flipper by \( k \) edges leading to the following algorithm (see also Figure 2).
Algorithm 2 Random $k$-Flipper

Choose random node $u_1 \in V$

for $i \leftarrow 1$ to $k + 2$ do

Choose random node $u_{i+1} \in N(u_i)$

for $i \leftarrow k + 2$ downto 2 do

if $\{u_i, u_{i+1}\} = \{u_{k+2}, u_{k+3}\}$ then $r \leftarrow i$

for $i \leftarrow 1$ to $r$ do

if $\{u_i, u_{i+1}\} = \{u_1, u_2\}$ then $\ell \leftarrow i$

if $r \geq \ell + 2$ and $\{u_{\ell}, u_r\}, \{u_{\ell+1}, u_{r+1}\} \notin E$ then

$E \leftarrow E \setminus \{\{u_{\ell}, u_{\ell+1}\}, \{u_r, u_{r+1}\}\}$

$E \leftarrow E \cup \{\{u_{\ell}, u_r\}, \{u_{\ell+1}, u_{r+1}\}\}$

---

Figure 2: The $k$-Flipper operation.

Due to space limitations we will not discuss all details of the Random $k$-Flipper operation here. Since a $k$-Flipper operation can simulate a 1-Flipper operation by using the first edge several times, the $k$-Flipper inherits most properties of the 1-Flipper. Unfortunately, this is no true for the uniform generation of $d$-regular connected undirected graphs. Nevertheless, carefully analyzing the effects of a Random $k$-Flipper operation to the edge boundaries of all subsets $S \subset V$ of size at most $|V|/2$ delivers the following theorem.

**Theorem 2** If we choose $d \in \Omega(\log n)$ applying $O(dn)$ Random $\Theta(d^2n^2 \log 1/\epsilon)$-Flipper operations transforms any given $d$-regular connected graph into a connected $d$-regular graph with expansion $\Theta(d)$.

So, the Random $k$-Flipper operation is able to build expander graphs very fast. Furthermore, we describe a simple scheme of how to deal with concurrent Flipper operations. This will further increase the convergence speed to expander graphs in case of the Random $k$-Flipper respectively the uniform probability distribution in case of the Random 1-Flipper.

### 3 Weighted Distributed Hash Tables

Karger et al. introduced the notion of Consistent Hashing, aka. Distributed Hash Table in [KLL+97]. Such a scheme gives a mapping from a set of data elements to a dynamic set of hosts. In the original setting data was to be distributed among different sets of hosts, so called views. The goal was to avoid swamped servers, by decreasing the usage of memory and to balance data fairly among the views.

For this, the hosts are mapped to some range using a hash function. Also the data elements are mapped to the very same range using an appropriate hash function. Now in every view (relevant
Figure 3: The working principle of consistent hashing, aka. distributed hash tables.

A sub-set of hosts) a data element is stored on a host, if this host’s image in the hash range minimizes the distance to the image of the data element (see Figure 3). If a sufficient number of copies of the hosts are mapped to the range, one can show that this leads to a fair balance of data elements [RLS+03, BCM02]. Now, if a new host is added to this system, then only data elements which will be stored on the new host need to be reassigned. This feature is called consistency.

Such distributed hash tables are universally applicable to many areas of distributed computing. First, they are introduced to distribute web sites among servers distributed around the globe relieving hot spots in the Internet [KLL+97]. Besides the area of Web Caching they are popular in Peer-to-Peer Networks, see CAN [RFH+01], Chord [SMK+01], Koorde [KK03], Viceroy [MNR02], Pastry [RD01], Tapestry [HKRZ02], and many more. A further important application field are Storage Area Networks (SAN), to overcome problems induced by huge RAID arrays. Here, the task is to distribute data on multiple heterogeneous disks that act like one virtual disk [BMMadHS+03].

3.1 State of the Art

We consider the more general case of weighted consistent hashing where compared to the original approach every host $v_i$ comes now with a positive weight $w_i$. Let $W = \sum_{i \in V} w_i$ be the overall weight. Then, the goal is to distribute a data element with probability $w_i/W$ to node $v_i$. Clearly, such a weighting is a helpful extension to all the application areas named above. Especially for Peer-to-Peer networks and Storage Area Networks such an extension is crucial. In Peer-to-Peer networks participants are not necessarily equipped with equal storage devices or transmission bandwidth. Hence, a weighting can improve performance.

The naive approach for introducing a weighted version of consistent hashing is to use $\left\lceil \frac{w_i}{\min_{j \in V} \{w_j\}} \right\rceil$ copies for each peer $w_i$. This is not feasible, if $\max_{j \in V} \{w_j\}/\min_{j \in V} \{w_j\}$ is too large. Furthermore, interesting nodes with small weights increases the number of copies of all nodes.

Brinkmann et al. presented a scheme to overcome this problem in [BSS02], but not as elegant as the original weighted consistent hashing and still uses a large number of copies. Furthermore, small disks are under-utilized and the scheme has to undergo special reorganization procedures if too many disks are in- or excluded.
3.2 New Results: Distributed Heterogeneous Hash Tables

In [DELIS-TR-0149] we introduced Distributed Heterogeneous Hash Tables (DHHT). DHHT is a generalization of the Consistent Hashing introduced by Karger et. al. in [KLL+97]. It uses a similar technique to map a dynamic set of documents \( D \) to a dynamic set of servers \( S \), by using a hash range \( M \ [0, 1) \), whereas the assignment of documents to servers strongly differs. In the original consistent hashing scheme \( S \) only consists of \( n \) homogeneous servers. Our approach [DELIS-TR-0149] overcomes this artificial restriction and adds to each \( s_i \in S \) a corresponding weight \( w_i \) that reflects the capacity of each server. This extensions allows to solve the following Problem.

Definition 3 The Heterogeneous Distribution Problem: Given a dynamic set \( S = \{s_1, \cdots, s_n \} \) of servers, a weighting function \( w : S \rightarrow \mathbb{R}^+ \), and a dynamic set of documents \( D = \{d_1, \cdots, d_m \} \). Find a mapping function \( f_{S,w} : D \rightarrow S \) with the following properties:

- Simplicity, which means, the function \( f() \) uses \( S, w, d \) as input and is calculated without the knowledge of \( D \) \{d\}
- Fairness, such that each server gets a comparative portion of data with respect to its weight:
  \[\forall u, v \in S, \frac{f_{S,w}^{-1}(v)}{w(v)} = \frac{f_{S,w}^{-1}(u)}{w(u)}, \text{ where } f_{S,w}^{-1}(s) := \{d \in D : f_{S,w}(d) = s\}\]
- Consistency means, if \(|S| \) or \( w(s) \) changes, the number of data reallocation steps that are needed to preserve fairness are minimal. In other words no unnecessary data movements!

The DHHT approach offers two different schemes, called the Linear Method and the Logarithmic Method, which both can solve the heterogeneous problem. For a detailed discussion of the different properties of both schemes, we would like to refer to the article [DELIS-TR-0149]. For easier comprehension the basic steps of how-to assign a document to a set of servers are explained here:

1. Choose for each \( s \in S \) a random position \( rs \) in \( M \) via hash function, \( rs = h(s) \)
2. Choose for a document \( d \in D \) a random position \( rd \) in \( M \) via hash function, \( rd = h(d) \)
3. Compute the height of the document \( d \) at the position \( rd \) for each server \( s \). Depending on the scheme use \( h(d) = ((rs - rd) \text{ mod } 1)/w \) for the Linear Method and \( h(d) = -\ln((1 - (rs - rd)) \text{ mod } 1)/w \) for the Logarithmic Method, where a \( \text{mod } 1 := a - [a] \)
4. Assign the document to the server which minimize the height of \( d \) at the position \( rd \)

The resulting mapping and the responsibilities of the hash interval for servers are illustrated in Figure 4. One can see how the capacity of servers and the length of the appropriate interval sections are correlating.

Compared to existing approaches like SHARE or SIVE [BSS02], which also try to solve the heterogeneous problem, the new models within the DHHT approach promises useful and none overhead causing benefits for peer-to-peer networks and storage area networks like:

- Coverage of the hash interval for data assignment is guaranteed since \(|S| \geq 1\) and is independent of joining or leaving servers.
- Direct and unique data assignment for new data or data reallocation caused by joining or leaving Servers.
- Capacity variations of servers during runtime are allowed and adjustable with minimal effort.

Actually the Logarithmic Method can also be used for homogeneous servers and heterogeneous documents, by exchanging the roles of Servers and Documents.
Corollary 1 Consider \( n \) same sized servers \( s_1, \ldots, s_n \) and \( m \) heterogeneous documents \( d_1, \ldots, d_m \). For all \( \epsilon > 0 \) and \( c > 0 \) there exists \( c' > 0 \), where we apply the Logarithmic Method with \( c'm \log n \) document fragments. Then with high probability, i.e. \( 1 - n^{-c} \), we achieve \( \epsilon \)-fairness.

4 Peer-to-Peer Networks Supporting Locality

In this section we briefly analyze existing peer-to-peer networks concerning their capabilities to support complex queries and several types of locality. Finally, we give a brief description of the 3nuts peer-to-peer network which supports complex queries and locality.

4.1 State of the Art: Brief Characterization of Existing Peer-to-Peer-Networks

Especially in the theory community peer-to-peer networking has been seen as a new opportunity to use sophisticated network graph designs for improving the network structure. So, communication network design techniques originated for linking processors of massively parallel computer networks were revisited. Examples are the \( k \)-dimensional torus for CAN (Content Addressable Networks [RFH+01]), the butterfly-graph for Viceroy [MNR02], the hypercube-based routing of Plaxton, Rajamaran and Richa [PRR97] for Pastry [RD01] and Tapestry [HKRZ02], DeBruijn-graphs for Koorde [KK03] and the Pagoda network [BKR+04], etc (see Table 1).

So, much emphasis was put in finding the most suitable graph structure for peer-to-peer networks, while little was thought about the distributed nature of this connection networks. So, many deterministic data structures are hard to maintain and can be easily attacked and this vulnerability to security threats and vicious denial-of-service or sybil attacks has been an inspiration to recent research developments.

4.1.1 Semantic Information in Peer-to-Peer Networks

In the first peer-to-peer networks either all information was based on a database allowing all kinds of semantic lookup (Napster) or they were distributed over the complete network (Gnutella [Gnu]) and were accessible only via network intensive flooding. Still, some semantic information was available since the users providing some information tend to gather information of similar concepts providing an owner-based semantic context information.

However, when the performance was improved by the use of hash functions any semantic interdependencies were sacrificed for this optimization objective. Peers stored lookup information without
Table 1: Characterization of existing peer-to-peer networks

<table>
<thead>
<tr>
<th>Network</th>
<th>Semantic</th>
<th>Backbone</th>
<th>Search Method</th>
<th>Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Napster</td>
<td>None (ownership)</td>
<td>Star</td>
<td>Server Query</td>
<td>Hybrid</td>
</tr>
<tr>
<td>Kazaa</td>
<td>None (ownership)</td>
<td>Hybrid Graph</td>
<td>Restricted BFS</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Gnutella</td>
<td>None (ownership)</td>
<td>Pareto Graph</td>
<td>BFS</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Chord</td>
<td>None (hashing)</td>
<td>Ring</td>
<td>Binary Search</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Koorde</td>
<td>None (hashing)</td>
<td>DeBruijn-Graph</td>
<td>DeBruijn-Routing</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>CAN</td>
<td>None (hashing)</td>
<td>$d$-dim. Torus</td>
<td>$d$-dim. Torus</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Pagoda</td>
<td>None (hashing)</td>
<td>DeBruijn-Graph</td>
<td>DeBruijn-Routing</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Viceroy</td>
<td>None (hashing)</td>
<td>Butterfly-Graph</td>
<td>Butterfly-Routing</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Pastry</td>
<td>None (hashing)</td>
<td>Mesh of Trees</td>
<td>Plaxton-Routing</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Tapestry</td>
<td>None (hashing)</td>
<td>Mesh of Trees</td>
<td>Plaxton-Routing</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Skipnet</td>
<td>Bottom-up</td>
<td>Ring</td>
<td>Skip-Lists</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>Brushwood</td>
<td>Top-Down</td>
<td>Tree</td>
<td>Skip-Lists</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>P-Grid</td>
<td>Top-Down</td>
<td>Mesh of Trees</td>
<td>Search-Tree</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>3nuts</td>
<td>Top-Down</td>
<td>Random Graph</td>
<td>Search-Tree</td>
<td>Peer-to-Peer</td>
</tr>
</tbody>
</table>

any semantic relation between them. Typical examples of such networks are Chord [SMK+01], Koorde [KK03], and Distance Halving [NW03a].

The peer-to-peer networks Skipnet [HDJ+02, NW03b] and Skip Graphs [AS03] overcome this problem. Semantic information can be encoded in a ring structure. Then a super-imposed search structure allows efficient lookup. In Skipnet the data need not to be distributed equally around this ring. A skip-list equivalent data structure allows efficient lookup. We call this approach a bottom up semantic approach, since when data is inserted in the bottom the search network on top needs to be re-established. In such bottom up networks changing the semantic structure produces non-trivial rebalancing and maintenance algorithms for the network.

Recently, some networks, like P-Grid [ACMD+03] and Brushwood [ZKW05], have been presented where semantics given by a search tree impose a tree-like network structures. New information are inserted from top into the network and therefore we call this approach a top-down network. This concept adds more flexibility in defining the data structure during the lifetime of the network.

4.1.2 Locality in Peer-to-Peer Networks

In Deliverable 6.2.1 of the DELIS project we pointed out three types of locality, which help improving performance and allow non-trivial queries. We now give short definitions of these three kinds of locality and then give an overview of our peer-to-peer network 3nuts which supports these kinds of locality. For a more detailed discussion of each of the locality types we would like to refer to Deliverable 6.2.1.

**Definition 4 (Network Locality)** A peer-to-peer network provides network locality if lookup operations can be performed with small latency.

This notion refers only to peers of the network. Often in peer-to-peer networking designers abstract from the underlying network (the Internet). Networks are constructed to provide small hop distance. Yet, a hop connecting computers in Greece and Australia counts as much a hop from one room to the next one of a building. Clearly, the locality of peers within the Internet has to be taken into
account for optimizing the network structure and the data lookup operations. There are few peer-to-peer networks supporting Network locality, e.g. Pastry [RD01], Tapestry [HKRZ02] and DHash++ [DLS+04].

**Definition 5 (Information Locality)** A peer-to-peer network provides information locality if closely related data elements are stored on network-wise close peers.

Distributed hash tables map data elements to (pseudo) random positions via the hash functions. Elements stored at a peer are completely unrelated. So, a structured query like a range query or a proximity search cannot be performed efficiently using a plain distributed hash table. Information locality is provided by most peer-to-peer networks which are not solely based on distributed hash tables, like Skipnet [HDJ+02, NW03b], Skip Grahs [AS03], P-Grid [ACMD+03] and Brushwood [ZKW05].

**Definition 6 (Interest Locality)** A peer-to-peer network provides interest locality, if peers can choose on providing lookup service and data storage for certain data. If peers choose to provide certain data, then the network structure allows efficient lookup to data relevant to a peer.

In the Web certain data is intrinsically local, e.g. most of all Polish web-sites are created in Poland and accessed from computers in Poland. Hence, it makes a lot of sense to store such data on Polish peers. Another aspect is that in peer-to-peer networks often data is published and distributed which inflicts with local law, e.g. violating copyright or containing offending contents. In a peer-to-peer network participants should be able to reject storing such data or even reject the assistance for looking up such data. So far there are no peer-to-peer networks supporting interest locality.

### 4.2 New Results: The 3nuts Top-Down Peer-to-Peer Network

We will now give a brief overview of the 3nuts peer-to-peer network, which is able to support network locality, information locality, and interest locality.

#### 4.2.1 Network Structure

To support complex queries — like range queries — 3nuts basic network structure is a search tree. We assume the structure of the search tree to be given by an external application. Of course, a search tree on its own is a rather weak structure for a communication network when implemented in the straightforward way with one peer representing one node of the tree. This would imply an extremely high load to the peer assigned to the root of the tree, make the root of the tree a single point of failure, etc.

Therefore, in Peanuts each of the tree nodes is represented by a constant degree random network. Simplified, the assignment of peers to tree nodes is solved as follows. We start with the root of the search tree which is represented by a random network consisting of all participating peers. Then, we recursively partition all peers of a search tree node into as many sets as there are child nodes of the search tree node. This recursive assignment is repeated until the a search tree node is represented by a single peer or the search tree node is a leaf of the search tree. Following this simplified scheme, a peer describes a path from the root to a leave of the search tree.

Besides links to neighbored peers of the random graphs, each peer maintains so called *shortcut links* at each level of the tree. This shortcut links are links to random peers participating in the search tree nodes the peer has not been assigned to. The shortcut links allow efficient routing in the network with $O(\log n)$ hops, where $n$ denotes the total number of peers in the network. Note, that this also holds in case of unbalanced trees (a proof can be found in [Abe02]).

An example of a search tree is given in Figure 5. The tree nodes describing the path of a specific peer $x$ is shown by orange colored nodes. The search tree nodes to which peer $x$ has shortcut links
are shown in yellow. The parts of the search tree shown by dashed lines are not visible for $x$, yet they can be reached by following shortcut links.

Note, that this combination of trees and random graphs is extremely useful since a random graph helps to overcome the shortcomings of the tree structure as a communication network and the tree structure allows efficient and complex queries, which would not be possible in random graph. Furthermore it is notable that following this scheme peers are assigned to the data and not vice versa. This allows us to preserve semantic information.

### 4.2.2 Load Balancing

It is very likely that the search tree will not be balanced if we are working with real world data. To manage unbalanced trees we use the DHHT scheme [DELIS-TR-0149] discussed in Section 3.2 of this deliverable. So, each subtree will get a fraction of peers which is proportional to the current weight of the subtree. Here, weight can be network load, storage capacity, etc., depending on the particular application. Note, that using the DHHT scheme we get a provable load balancing (see Corollary 1) instead of the heuristics used in many other peer-to-peer networks.

Figure 6 shows the peer to subtree assignement as it would take place in the root of the search tree shown in Figure 5. There, each peer is assigned to one of the subtrees representing Music, Movies, or Documents.

### 4.2.3 Network Maintenance

3nuts only uses very simple and local update rules to maintain the network structure. This maintenance enfolds maintenance of the random graphs representing the tree nodes and adapting the tree structure in case of dynamic trees. The random graphs are maintained by Flipper operations [DELIS-TR-0148] described in Section 2.3 of this deliverable. This guarantees, that all these graphs are truly random and therefore make the network structure robust and reliable under the possible churn of a highly dynamic peer-to-peer network. Furthermore, 3nuts inherits the expansion property of random graphs what in turn allows efficient broadcast within tree nodes using randomized rumor spreading [KSSV00].

The maintenance operations concerning the update of the tree structure are integrated into the Flipper protocol and therefore do not introduce additional network traffic. Whenever two peers
meet during a Flipper operation, they exchange information about the tree structure. Due to the expansion property new information spreads quickly.

4.2.4 Locality in 3nuts

The 3nuts peer-to-peer network supports network locality, information locality as well as interest locality. For network locality, a peer $x$ periodically verifies its shortcut links and asks them for some of their neighbors. If one of these neighbors appears to be latency wise closer to $x$ than the old shortcut link, the shortcut link is replaced. Due to the Flipper operations performed in the random networks of the shortcut link, the neighborhood of the shortcut link is rapidly mixing. Once again the expansion property is extremely helpful here. Information locality is supported directly by the use of a search tree, which is able to group related documents (as indicated in Figure 5). Finally, 3nuts supports interest locality. For this we allow a peer to volunteer for certain parts of the tree besides the path it has been assigned to by the DHHT scheme. This means, that a peer can be assigned to several paths of the tree, thus introducing additional load to the peer. However, keep in mind that this additional load is voluntary and in return the peer will have a faster lookup for documents related to the part of the search tree it volunteers for.

5 Result Dissemination

The research in this Working Package has led to the following publications on international conferences [DELIS-TR-0148, MS05, DELIS-TR-0149, DELIS-TR-0287, DELIS-TR-0308]. These publications are available on request from the coordinating site UPB. Other publications [MS06] are in preparation and will be published by the beginning of 2006. Some results described in this document have also been presented at:

- the DELIS SP6 meeting in Saarbrücken, July 19th-20th 2005
- the Dagstuhl seminar "Algorithmic Aspects of Large and Complex Networks", Dagstuhl, Germany, September 2005
- the third Workshop on Random Graphs and Randomized Algorithms in Bertinoro, Italy, June 20th-24th 2005
• the Meeting of the EU COST Action 295 (joint session with DISC 2005) in Krakow, September 29th-30th, 2005
• the Workshop on Stable Network Structures in Dynamic Systems in Tübingen, Germany, December 20th-21st 2005.

References


Gnutella. The gnutella protocol specification v0.4.


JXTA. The jxta peer-to-peer networking suite.


