Technical University of Vienna
Information Systems Institute
Distributed Systems Group

# Gridella: A self-organizing P2P system

Karl Aberer, Manfred Hauswirth,
Magdalena Punceva and
Roman Schmidt
karl.aberer@epfl.ch
M.Hauswirth@infosys.tuwien.ac.at
magdalena.punceva@epfl.ch
R.Schmidt@infosys.tuwien.ac.at

TUV-1841-01-09 Oct 26, 2001

*This paper describes our Gnutella compatible P2P system Gridella which is based on the P-Grid approach. P-Grid has several advantages over the standard Gnutella infrastructure, for example, that probabilistic estimates can be given for successful search requests and that it scales gracefully in the total number of nodes and data items. Gridella is designed for interoperability with Gnutella and the long-term goal is to replace the existing Gnutella infrastructure with Gridella to make it more self-organizing and get around known problems of Gnutella. On the basis of an analysis of Gnutella we describe the P-Grid approach and the mapping algorithm that form the algorithmic foundations of Gridella. We present Gridella's architecture and communication patterns and compare it with Gnutella.*

Keywords: Peer-to-peer, self-organization, distributed information systems



Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

# Gridella: A self-organizing P2P system

Karl Aberer*, Manfred Hauswirth†, Magdalena Punceva*, Roman Schmidt†

**Abstract**

This paper describes our Gnutella compatible P2P system Gridella which is based on the P-Grid approach [1]. P-Grid has several advantages over the standard Gnutella infrastructure, for example, that probabilistic estimates can be given for successful search requests and that it scales gracefully in the total number of nodes and data items. Gridella is designed for interoperability with Gnutella and the long-term goal is to replace the existing Gnutella infrastructure with Gridella to make it more self-organizing and get around known problems of Gnutella. On the basis of an analysis of Gnutella we describe the P-Grid approach and the mapping algorithm that form the algorithmic foundations of Gridella. We present Gridella's architecture and communication patterns and compare it with Gnutella.

## 1 Introduction

The limitations of client/server-based systems become evident in an Internet-scale distributed environment: Resources are concentrated on a small number of nodes which must apply sophisticated load-balancing and fault-tolerance algorithms to provide users with continuous and reliable access. Additionally, network bandwidth to/from successful Internet servers must be increased steadily because no "a priori" caching and replication strategies exist. These concepts were introduced "a posteriori" when the WWW as the most successful Internet service developed into a network bandwidth nightmare. A considerable amount of scientific work now tries to remedy these problems.

For several application domains peer-to-peer (P2P) systems offer an alternative to traditional client/server systems: Every node (peer) of the system acts as a client and server (servent) and provides part of the overall information available from the system. Each peer "pays" its participation by providing access to its computing resources. P2P systems can be characterized by the following properties:

*Distributed Information Systems Laboratory, Communication Systems Department, Swiss Federal Institute of Technology, Lausanne (EPFL), 1015 Lausanne, Switzerland, tel: +41-21-693-4679, fax: +41-21-693-8115, {karl.aberer, magdalena.punceva}@epfl.ch

†Distributed Systems Group, Information Systems Institute, Technical University of Vienna, Argentinierstr. 8/184-1, A-1040 Austria, tel: +43-1-58801-18417, fax: +43-1-58801-18491, {M.Hauswirth, R.Schmidt}@infosys.tuwien.ac.at

- no central coordination
- no central database
- no peer has a global view of the system
- global behavior emerges from local interactions
- all existing data and services should be accessible
- peers are autonomous
- peers and connections are unreliable

This intriguing approach circumvents many problems of C/S systems in a very simple fashion but pays for this simplicity with considerably higher complexity for searching, node organization, security, etc. For example, Napster [14], which made the P2P idea popular, gets around some of the complexity by employing a centralized database holding references to files on peers. Thus it is not a "pure" P2P system according to the above properties. Gnutella solves this and other problems, but at the cost of using a communication-intensive search mechanism. Many P2P systems are based on very simple approaches which are easy to deploy but suffer from problematic shortcomings while a considerable amount of research in the areas of distributed and cooperative information systems is not exploited by the P2P community yet.

In this paper we present Gridella, our Gnutella compatible P2P system that is based on the P-Grid approach [1]. We wanted Gridella to be compatible with Gnutella to "infiltrate" the existing Gnutella infrastructure and gradually replace standard Gnutella nodes with Gridella nodes. Gridella can communicate with existing Gnutella nodes but can offer advanced services when communicating with other Gridella nodes.

In Section 2 we describe Gnutella's underlying architecture and communication patterns to understand its benefits and problems which motivated the development of Gridella. This is followed by a presentation of Gridella's essential algorithmic foundations: P-Grid (Section 3), the process of constructing the P-Grid (Section 4), and a mapping scheme for search keys (Section 5). Section 6 then provides some details of Gridella's architecture and interaction patterns and compares it with Gnutella. We relate Gridella to other approaches in the field in Section 7 and end the paper with our conclusions in Section 8.

## 2 Gnutella

The Gnutella system is a decentralized file-sharing system whose participants form a virtual network communicating in a P2P way via the Gnutella protocol [6] which is a simple protocol for distributed file search. To participate in Gnutella a peer first must connect to a known Gnutella host (specialized servers return lists of hosts to get started; this is outside the Gnutella protocol specification). The protocol consists of 5 basic message types shown in Table 1.

These messages are routed by all servents using a constrained broadcast mechanism: Upon receipt of a message the servent decrements the message's time-to-live (TTL) field. If the TTL is greater than 0 and it has not seen the message's identifier before (loop detection) it resends the message to all the

| Type | Description | Contained Information |
|---|---|---|
| Ping | Announce availability and probe for other servents | None |
| Pong | Response to a Ping | IP address and port number of the responding servent; number and total kB of files shared |
| Query | Search request | Minimum network bandwidth of responding servent; search criteria |
| QueryHit | Returned by servents that have the requested file | IP address, port number, and network bandwidth of responding servent; number of results and result set |
| Push | File download requests for servents behind firewalls | Servent identifier; index of requested file; IP address and port to send file to |

Table 1: Message types in the Gnutella protocol

peers it knows. Additionally the servent checks whether it should respond to the message: If it receives a *Query* message it checks its local file store and if it can satisfy the request, responds with a *QueryHit* message. Responses are routed along the same path as the originating message.

A simplified Gnutella "session" works as follows: Servent $A$ connects to servent $B$ and sends a *Ping* message. $B$ responds with a *Pong* and forwards the *Ping* as described above (e.g., it forwards it to its peers $C$ and $D$ who respond with another *Pong*, and so on). After some time $A$ knows a number of servents and vice versa. It routes messages as described above and may initiate queries. When it receives a *QueryHit* for one of its queries it tries to connect directly to the servent specified in the *QueryHit* and runs a simplified HTTP GET interaction to retrieve the file. In case the requested servent is behind a firewall it may send a *Push* message (along the same way as it received the *QueryHit*) to the firewalled servent. The *Push* message specifies were the firewalled servent can contact the requesting servent to run a "passive" GET session. If both servents are behind firewalls then the download is impossible.

## 2.1 Discussion of Gnutella

From a user's perspective Gnutella is a simple yet effective protocol: Hit rates for search queries are reasonably high, it is fault-tolerant towards failures of servents, and adopts well to dynamically changing "peer populations." However, from a networking perspective, this comes at the price of very high bandwidth consumption: Search requests are "broadcast" over the network and each node receiving a search request scans its local database for possible hits.

For example, assuming a typical TTL of 7 and an average of 4 connections $C$

per peer (i.e., each peer forwards messages to 3 other peers) the total number of messages originating from one Gnutella message (including the responses) can be calculated as:

$$2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26240 \quad (1)$$

Recent experiments [19] showed that in a real-world setting this accumulates up to 3.5Mbps (or 353,396 queries in 2.5 hours as another experiment's result in [19]). To remedy this [19] suggests to apply caching which reduces the traffic considerably.

Also, no estimates on the durations of queries and no probability for successful search requests can be given. Very little is known about the topology of the Gnutella network which would provide the foundation for an accurate mathematical model and would aid the development of efficient algorithms that exploit structural properties. As a first step a recent study [11] investigated Gnutella's topology and has shown that it exhibits strong small-world properties [12] and a power law distribution of node degrees.

Besides these technical problems non-technical ones such as "free riding" challenge Gnutella [3]. "Free riding" means that most Gnutella users do not provide files to share and if, only a very limited number is interesting. [3] shows that nearly 70% of Gnutella users share no files and nearly 50% of all responses are returned by the top 1% of the sharing hosts. This social problem starts to transform Gnutella into a C/S like system which soon may have to face the technical (degradation of performance, vulnerability, etc.) and legal problems of Napster.

Another "social" issue which is not addressed by Gnutella is reputation: In a P2P system peers frequently have to "meet" unknown peers and have no possibility to judge their reputation, i.e., to what extent they can trust the peers and the data provided by them. As shown in [2] Gridella's P-Grid approach may also be used to address this issue efficiently.

These shortcomings of the Gnutella motivated the development of the P-Grid-based Gridella system described in the following sections.

# 3 The P-Grid Approach

The underlying idea of the P-Grid approach [1] is to create a virtual binary search structure with replication that is distributed over the peers and supports efficient search, i.e., search time and number of generated messages grow $O(\log_2 n)$ with the number of data items $n$ in the network. This approach is comparable to other approaches such as [10, 15, 22] to construct scalable, tree-based, distributed indexing structures. However, a main innovation is that the construction and the search/update operations can be performed without any central control and/or global knowledge in an unreliable environment. This is achieved by applying randomized, distributed algorithms. All algorithms for

creating and using the search structure are realized through completely decentralized cooperation among the peers. Consequently this search structure exhibits the following properties:

- it is completely decentralized;
- all peers serve as entry points for search;
- interactions are strictly local;
- it uses randomized algorithms for access and search;
- probabilistic estimates for the success of search requests can be given;
- search is robust against failures of nodes; and
- it scales gracefully in the total number of nodes and data items.

## 3.1 The P-Grid Search Structure and Algorithm

P-Grid is a virtual distributed binary search tree which is distributed among a community of peers. This means that each peer only holds part of the overall tree which comes into existence only through the cooperation of the individual peers. The position of every participating peer is determined by its "path," i.e., the binary bit string representing the subset of the overall information in the tree that the peer is responsible for. For example, the path of *peer4* in the example P-Grid shown in Figure 1 is '10' which means that it is responsible for all data items whose key begins with '10', i.e., stores them.

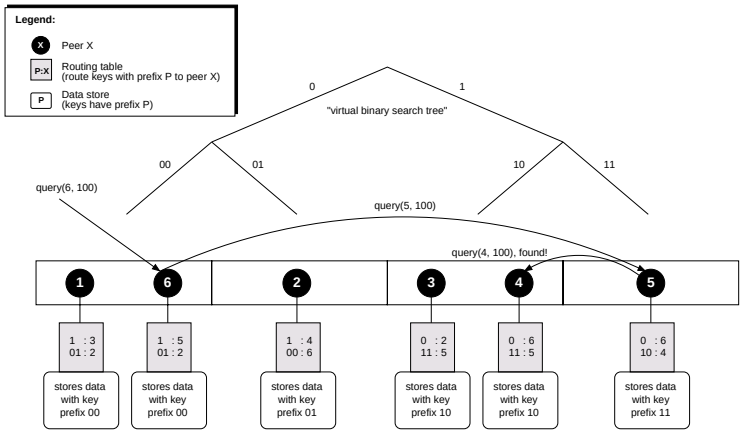


Figure 1: Example P-Grid

The paths implicitly partition the search space and define the structure of the virtual binary search tree. Its construction will be explained in the next section. As can be seen from Figure 1 multiple peers can be responsible for the same path. For example, *peer1* and *peer6* are both responsible for keys beginning with '00'. Such replication is up to the individual peers and improves the robustness and responsiveness of the P-Grid since we assume that peers are not online all the time but with a certain, possibly low probability.

As each peer should be able to serve as entry point for any search query the peers must also store routing information. This means that if a peer is presented with a binary query string and cannot satisfy the query itself it must forward the query to a peer which is "closer" to the result. P-Grid's routing approach is simple but efficient: For each bit in its path each peer stores the address (IP address + port number; 1 computer can host multiple peers) of at least one other peer who is responsible for the other side of the binary tree at this level.

For example, *peer1* will forward queries starting with '1' to *peer3* which is in *peer1*'s routing table and whose path starts with '1'. *Peer3* may either be able to satisfy the query or forward it to another peer depending on the following bits of the query. If *peer1* gets a query starting with '0' then it could be responsible since its path also starts with '0'. To decide this, *peer1* looks at the next bit of the query and if it is '0' it is responsible for the query. Otherwise it will check its routing table and forward the query to *peer2* whose path starts with '01' which matches the prefix of the query. Figure 1 shows the routing tables as grey boxes.

The P-Grid construction algorithm which will be explained in the next section guarantees that the routing tables of the individual peers are constructed in a way that there always exists at least one path between any two peers of a P-Grid which means that any query can be satisfied regardless which peer is queried. This property must be seen "relative" to replication: For example, in Figure 2 that visualizes this property for the P-Grid of Figure 1, no path between *peer3* and *peer1* exists, but there is a path from *peer3* to *peer6* which holds the same data as *peer1*.

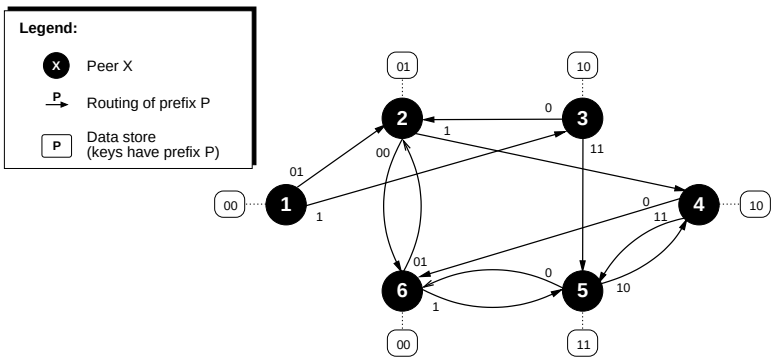


Figure 2: Example P-Grid network

A search request in P-Grid works as follows: The query is sent to an arbitrary peer. In Figure 1 a query for '100' is sent to *peer6*. Since *peer6* is responsible for keys starting with '00' it checks its routing table for the longest common prefix with the query which is '1' and forwards the query to *peer5* that is given in the routing table. In a real setup multiple peers would be listed for each prefix in the routing table and the peer to forward the query to would be chosen

randomly. Without constraining general applicability we assume in this simple example that each prefix is "serviced" by one peer entry in the routing table.

Upon receiving the query *peer5* that is responsible for prefix '11' does the same checks as *peer6* before and finds out that the query is to be forwarded to *peer4* which is the longest common prefix in *peer6*'s routing table. *Peer4* in turn has no longer common prefix in its routing table so it is clear that it must search in its local data store for data with the key '100'. If the key exists a reference to the associated data is returned to the original requester *peer6* that may then request the data. This example demonstrates that the order of the search is equivalent to a binary tree search regardless of the entry point of the query.

The algorithm to process a query is shown in Figure 3. The parameter `peer` indicates the address of the peer to send the query to, `query` is the search string, and `index` indicates the progress of the search, i.e., how many bits of the query have already been processed. Initially `index` is 0.

`sub_path(string, from, to)` returns the substring of `string` that starts at position `from` and ends at position `to`. `common_prefix_of(str1, str2)` returns the common prefix of the strings `str1` and `str2`. `get_refs(index)` returns the list of addresses in the routing table for a prefix of length `index`. `random_select(refs)` returns an address from this list and removes it from `refs`. `online(ref)` returns true if the referenced peer is online.

```
1     search (peer, query, index) {
2        found = NULL;           /* found: the address of the responsible peer */
3        rempath = sub_path(path(peer), index+1, length(path(peer)));
4        compath = common_prefix_of(query, rempath);
5        IF length(compath)=length(query) OR length(compath)=length(rempath) THEN
6           found = peer;
7        ELSE
8           IF length(path(peer)) > index + length(compath) THEN
9              new_query = sub_path(query, length(compath) + 1, length(query));
10             refs = get_refs(index + length(compath) + 1);
11             WHILE |refs| > 0 AND NOT found
12                ref = random_select(refs);
13                IF online(ref)
14                   found = search(ref, new_query, index + length(compath));
15       RETURN found;
16    }
```

Figure 3: P-Grid search algorithm

The algorithm first compares the common prefix of the peer's path and the query submitted. As the query string could already be truncated by the first `index` bits, the path of the peer must also be adapted. This is an optimization because at level `index` of the virtual search tree the equality of the first `index` bits is guaranteed. Only the following bits are relevant (line 3) and must be compared with the query to find their common prefix (line 4). If the common path (`compath`) is as long as the query or the remaining path then the peer that is responsible for this query was found (line 5).

Otherwise the query must be forwarded. This is only possible if the peer is specialized sufficiently (line 8): The common prefix is stripped off the query

(line 9), the routing table is queried for the list of peers to forward the query to (line 10) and then the remaining query `new_query` is forwarded recursively to a random peer from this list (if it is online) until the list is exhausted or the search has succeeded (lines 11–14).

# 4 P-Grid Construction

Having introduced the access structure and the search algorithm an important question remains: How is the P-Grid to be constructed? As there exists no global control this has to be done by local interactions only. The idea is that whenever peers meet, they refine the access structure. Peers may meet randomly because they are involved in other operations or because they systematically want to build the access structure. But assuming that by some mechanisms they meet frequently the process works as follows.

Initially, all peers are responsible for the whole search space, i.e., all search keys. So when two peers meet they split the search space into two parts and each one takes over responsibility for one half and stores the other's address to cover the other part of the search space. The same happens whenever two peers meet that are responsible for the same path.

In the course of the P-Grid construction additional cases will occur: If peers meet whose paths share a common prefix they can initiate new exchanges by forwarding each other to peers they are referencing themselves. If the meeting peers' paths are in a prefix relationship the peer with the shorter path can specialize by extending its path. To obtain a balanced P-Grid it will specialize in the opposite way the other peer has already done at that level. Figure 4 shows the complete construction algorithm in pseudo code.

Simulations [1] show that this algorithm has the following properties:

- The convergence speed to a complete P-Grid is (practically) not dependent on the total number of peers, i.e., each peer participates in a constant number of exchanges independent of the population size.
- The algorithm scales gracefully with growing maximum path length.
- For obtaining fast convergence the maximum allowed recursion depth should exceed a minimum value (e.g., 2 for paths of length 6).

These results show that P-Grids can be constructed efficiently in a self-organizing system without central control. Additionally, the bootstrap algorithm can be considered as a uniform, self-stabilizing, distributed algorithm [18].

In [1] it was also shown that the number of peers responsible for the same keys is distributed uniformly with a low deviation from the expected average number of peers responsible for a key. In the next section we will give an algorithm that supports the uniform distribution of the data over the binary keys, such that the workload will be evenly distributed among the peers.

Data updates in the P-Grid require the identification of all replicas which stored the concerned data. [1] shows that the best strategy to identify them is a breadth-first search with limited recursion breadth (e.g., 2). Still not all replicas will be found with this simple approach in general. To compensate for this, the

```
1     exchange(a1, a2, r)  {
2        determine the common prefix of path(a1) and path(a2) and its length lc;
3
4        exchange references at the level where the paths match;
5        li = length of remaining path of ai;
6        /* Case 1: both paths empty, introduce new level */
7        CASE l1 = 0 AND l2 = 0 AND lc < maximum possible path length
8           extend path(a1) with 0 and path(a2) with 1;
9           add mutual references for future search;
10       /* Case 2: one remaining path empty, split shorter path */
11       CASE l1 = 0 AND l2 > 0 AND lc < maximum possible path length
12          extend path(a1) by one bit different to the corresp. bit in path(a2);
13          update references of a1 with a2;
14       /* Case 3: analogous to case 2 with roles exchanged */
15          ...
16       /* Case 4: recursively exchange with referenced peers */
17       CASE l1 > l2 > 0 AND r < maximum recursion depth
18          take a reference from a2 at the level of the common prefix;
19          a1 performs a new exchange with the referenced peer
20         (which shares with a1 a longer common prefix);
21       /* Case 5: analogous to case 2 with roles exchanged */
22             ...
23    }
```

Figure 4: P-Grid construction algorithm

query strategy performs multiple searches on the P-Grid and then decides which result is the correct one based on the majority of identical answers. Thus a high probability of correctness of an answer can be achieved.

# 5 Mapping File Names into Binary Keys

In the P-Grid approach we assume that search keys have a binary representation and are uniformly distributed. Both assumption, however, do not hold for real filenames. Thus we provide a mapping scheme that calculates a binary representation from a filename string. To support search on these binary keys the mapping has to satisfy a prefix property for strings $s_1$ and $s_2$:

$$s_1 \; prefix \; s_2 \Rightarrow key(s_1) \; prefix \; key(s_2).$$

The construction algorithm is given in Figure 5.

The algorithm proceeds by first constructing a balanced trie structure based on a sample database of search strings. The trie structure is then used to compute the binary keys for search keys. This mapping function is computed based on a sample database that is constructed by systematically collecting filenames from the current Gnutella system. Our assumption is that if the sample database is large enough, it provides a good approximation of the global distribution of filenames such that the mapping constructed for the sample database results in a reasonable uniform distribution of key values for all filenames. In Section 5.1 we provide experimental evidence that this statement is valid. Thus we package

```
1     MakeTrie(sampledb) {
2        /* sort the sampledb in lexicographical order */
3        SortLex(sampledb);
4        /* find the common prefix for all the strings from sampledb */
5        commonprefix = CommonPrefix(sampledb);
6        /* We choose the middle string from sampledb and take the prefix of
7           length commonprefix+1, which is enough to split the sampledb into two
8           approximately equal parts. It is possible to explore different
9           alternatives at this point, in order to achieve a more balanced split */
10       if Size(sampledb) > MaxLeafStore then
11          mid = Prefix(sampledb[Quotient(Size(sampledb), 2)],
12                       Length(commonprefix) + 1);
13       for j = 1 to Size(sampledb) do
14           if sampledb[j] is lexicographically smaller than mid then
15              lowpart = Append(lowpart,sampledb[j]);
16           if sampledb[j] is lexicographically greater than mid then
17              highpart = Append(highpart,sampledb[j]);
18       if Size(lowpart) > MaxLeafStore then
19          left = MakeTrie(lowpart)
20       else
21          left = null;
22       if Size(highpart) > MaxLeafStore then
23          right = MakeTrie(highpart)
24       else
25          right = null;
26       root = mid;
27       TrieSet(root, left, right);
28    }
```

Figure 5: Trie construction algorithm

the mapping scheme and the sample database with the other P-Grid software components.

The trie building algorithm takes a sample search string database `sampledb` as parameter that contains unique strings of length `len` which are substrings of actual search strings. First `sampledb` is sorted in lexicographical order and then split into two approximately equally sized parts by taking the shortest possible string such that the lower part contains the strings that are lexicographically smaller than the string and the higher part contains the strings that are lexicographically greater. Then the value of this shortest string is stored in the root of the tree and the function is called recursively for both parts which construct the left (lower part) and right (higher part) branch of the trie. The splitting proceeds until there are less or equal than `MaxLeafStore` strings in the database.

By using this trie strings (filenames) can be mapped into binary keys as shown in Figure 6.

The binary key is calculated in the following way: The string is lexicographically compared to the root value of the trie and if it is prefix of or equal to the root then the calculation terminates returning the binary key; otherwise if it is smaller, then '0' is appended to the key and the function is called recursively with the left subtree; if it is greater '1' is appended to the key and `FindKey` is called with the right subtree.

```
1     FindKey (trie, filename) {
2        key = {};
3        if (trie = null) or (filename is prefix or equal to trie.root) then
4           return key;
5        else
6           if filename is lexicographically smaller than trie.root then
7              key = Append(key, 0);
8              key = Append(key, FindKey(trie.left, filename);
9           else
10             key = Append(key, 1);
11             key = Append(key, FindKey(trie.right, filename);
12    }
```

Figure 6: Mapping strings into binary keys

## 5.1 Experiments

We modified the open source Gnutella client Furi [21] to log all queries that were routed through it and used this data to construct a large database of Gnutella queries to evaluate the quality of the mapping algorithm. From the query database we derived a sample database that we used for constructing the trie structure. Then we tested the trie structure by encoding the complete set of search strings and verifying that the resulting keys were distributed uniformly. We give one exemplary result to illustrate this: Of 33799 logged search strings of length 4 we randomly selected 1951 strings for the sample database. We used this set to construct the trie using $MaxLeafStore = 30$ which resulted in 99 different keys. When generating the keys for all search strings, the maximum number of strings mapped to one key is 798. A perfectly uniform distribution would result in a maximum of 342 search strings per key such that in the worst case slightly more than twice the number of search strings are encoded into the same key as would be with a perfectly uniform encoding. Thus the resulting distribution is of fairly good quality with respect to uniformity and the workload for peers for storing data and answering queries will be distributed approximately uniformly as well. This is also illustrated by the frequency histogram shown in Figure 7. It shows that we achieve almost a normal distribution.

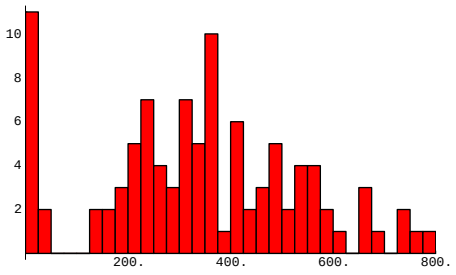


Figure 7: Key frequencies in steps of 25

# 6 The Gridella System

Gridella is our P-Grid-based, Gnutella compatible P2P system that targets efficient distributed search, reduction of network traffic, reusable components, and multi-protocol support. Gridella is written in Java and will be released under the GNU General Public License.

## 6.1 Component and Communication Model

Figure 8 shows the main components of the Gridella system.

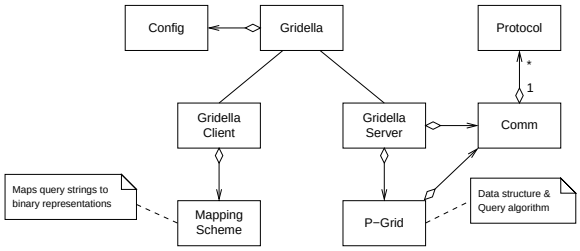


Figure 8: Gridella core system components

The Gridella client provides all user-related functionality so that an arbitrary GUI may be used while the Gridella server handles data management and communication. The communication subsystem provides communication abstractions for inclusion of arbitrary protocols (at the moment we support the Gnutella and Gridella protocols) which allows Gridella to run on top of arbitrary protocols.

Figure 9 shows a typical search interaction between two Gridella peers.

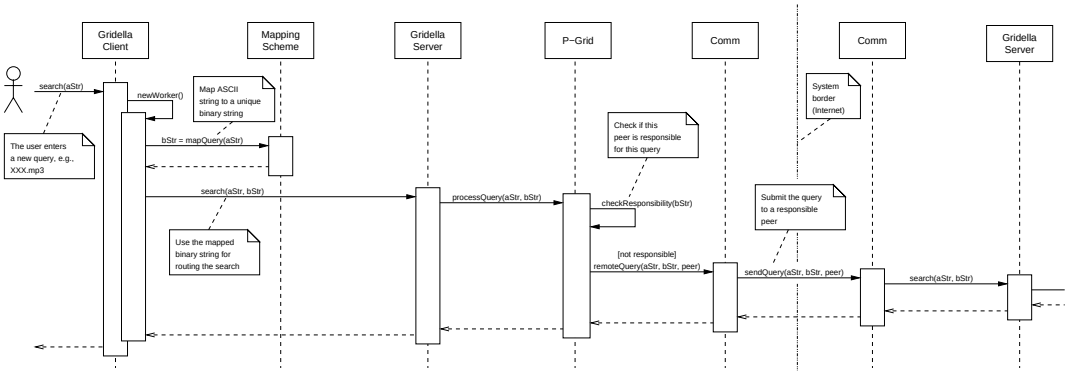


Figure 9: A Gridella interaction

If the user enters a query it is mapped into the binary P-Grid representation (Section 5) and sent to the local Gridella server that checks whether it is responsible. If yes, the requested data is returned. Otherwise it determines which peer to contact (Section 3) and forwards the query by using the services of the communication subsystem. At the recipient the query is forwarded to the Gridella server and the same pattern is repeated. The other interactions, for example meeting of two peers, announcing a peer's availability, or downloading data are straight-forward. Also interacting with non-Gridella peers is simple. For example, if the recipient is a Gnutella peer, the mapping step is skipped and the Gridella peer would forward the original query via the Gnutella protocol.

## 6.2 Comparison with Gnutella

Gridella can be viewed as a layer on top of Gnutella that provides additional abstractions and functionality: Gnutella provides the basic communication and Gridella adds directed, efficient search due to its underlying P-Grid approach. This is superior to Gnutella which searches in a trial-and-error way and thus requires considerably more network bandwidth.

In an analytic performance study we compared the number of messages required to find a specific data item in Gridella and Gnutella. We consider peer populations between 20.000 and 200.000 peers that are online with a probability of 0.3. Our goal was to achieve a search success probability of 0.99. For Gridella we assumed that each peer stores 1000 data items which is necessary to determine the key length.

To achieve a search success probability of 0.99 in Gnutella we can show that it is sufficient to create 22 replicas of each data item and to have a search horizon of 70% of all Gnutella peers, i.e., to reach 70% of the Gnutella population with a search message. Based on this we can compute the required number of search messages (Formula 1 in Section 2.1). For Gridella we determined the number of messages required to traverse the Gridella search structure in the worst case. The results are shown in Table 2.

| peers | Gridella messages | Gnutella messages |
|---|---|---|
| 20000 | 61 | 8744 |
| 40000 | 63 | 26240 |
| 60000 | 65 | 26240 |
| 80000 | 65 | 78728 |
| 100000 | 68 | 78728 |
| 120000 | 69 | 78728 |
| 140000 | 68 | 78728 |
| 160000 | 69 | 78728 |
| 180000 | 69 | 78728 |
| 200000 | 72 | 78728 |

Table 2: Performance comparison of Gridella and Gnutella

The variations for Gridella are the result of the simulation of the probabilistic process simulating Gridella's search algorithm. The steps in the results for Gnutella correspond to the steps in the time-to-live which must be incremented for higher numbers of peers to meet the 70% requirement. The results clearly demonstrate the benefit of using an access structure even if we have to take into account some modest storage demand and update overhead in Gridella.

As has been shown in [1] P-Grid offers several additional advantages which can be exploited by Gridella: It provides analytical proofs on bounds for reliability of answering search requests, fault-tolerance is high, and the structure of the overall system can be configured flexibly.

### 6.3 Future Work

We want to include reputation and trust into Gridella based on the approach proposed in [2]. This approach relies on P-Grid so that the existing implementation can be reused. The natural next step then is to address security issues such as authenticity and confidentiality of information. With these improvements P2P would be an interesting environment for new e-commerce models.

Also we intend to address the free-riding problem by introducing economic concepts, where users have to "pay" for the services they use. This does not necessarily mean the exchange of monetary values but a market-driven approach where a micro-payment system with an artificial currency could be used to balance requests with offers as suggested by [13, 20]. Offers and downloads from a peer would earn the peer credits which in turn it could use for paying for services it requests from other peers.

## 7 Related Work

A number of approaches are described in the literature that address scalable, decentralized access schemes. All of them differ from our approach because they exclusively allow exact search for file (or object) identifiers rather than text-based search for filenames. Also the autonomy of peers is normally severely limited with respect to the kind of search requests that they have to support whereas our approach supports considerable freedom.

Several approaches for serverless distributed file systems exist [4, 5, 17] that provide secure file storage in an insecure environment. They all encrypt the information to be stored and devise mechanisms to distribute and retrieve files in an unreliable environment. As our approach they have the problem of requiring a self-organizing routing structure and as far as published their routing approaches bear similarity with P-Grid. In contrast to Gridella they do not focus on information sharing and search, which motivated and influenced the Gridella architecture and search capabilities, i.e., allowing simple content-based search on filenames rather than supporting search for (possibly encrypted) file identifiers only.

Similarly, several architectures for decentralized lookup services have been published [7, 15, 16] that all devise scalable access structures to search for identifiers stored in distributed directories. In particular [15] uses virtual binary search trees similar as P-Grid. These approaches differ in two respects from our work: They only consider the problem of equality search for keys and make some assumptions that mildly violate the autonomy of the peers, such as fixed roles of the peers that are globally assigned, either by virtue of the IP address [7] or by a globally generated random assignment [15], or the requirement of a shared list of bootstrap entries to the network [16], which are avoided by employing the P-Grid bootstrap algorithm.

Besides these related approaches there is on-going work to increase the interoperability of P2P systems and to define a universal architecture for P2P systems. JXTA [8] defines a three layer P2P software architecture, a set of XML-based protocols, and a number of abstractions and concepts such as peer groups, pipes, and advertisements to provide a uniform platform for applications using P2P technology and for various P2P systems to interact. The first version of JXTA has been released recently and we are evaluating it for Gridella.

# 8 Conclusions

By popularizing the P2P approach in simple yet very successful and influential systems such as Napster and Gnutella the "Internet Community" has proven again its incubator capabilities for revolutionary systems and has somewhat "out-performed" the scientific community. A similar thing has occured some time ago with the introduction of the World-wide Web which boosted the Internet into everyday life. Although such developments are very helpful for spreading and advancing new technologies, now we know that this often comes at the cost of lacking scientific foundations which impedes development in the long run.

At the moment we still have the opportunity to put P2P systems on firm scientific foundations by combining state-of-the-art methodological and engineering know-how and thus advancing this paradigm. The work presented in this paper is a first step in this direction. It improves the highly chaotic and inefficient Gnutella infrastructure with directed search and advanced concepts which enhance efficiency and provides a consistent mathematical model for further reasoning and research. Still a considerable amount of research and experiments will be necessary to make P2P systems feasible for application domains beyond mere MP3 and image exchange, for example as a new paradigm for decentralized e-commerce systems, or for new types of network infrastructures such as mobile ad-hoc networks [9].

## References

[1] Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Proc. International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[2] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. Technical report DSC/2001/029. Swiss Federal Institute of Technology, Lausanne (EPFL), 2001.

[3] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. Technical report. Xerox PARC, 9 September 2000.

[4] William J. Bolosky et al. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *Proc. International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.

[5] Ian Clarke et al. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability.* LNCS 2009, Springer, 2001.

[6] Clip2. *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, 15 June 2001. http://www.clip2.com/GnutellaProtocol04.pdf.

[7] Frank Dabek et al. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. *Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[8] Li Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, **5**(3), May/June 2001.

[9] J. P. Hubaux et al. Towards self-organized mobile ad-hoc networks: the Terminodes project. *IEEE Communications Magazine*, January 2001.

[10] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, **9**(3), May/June 1997.

[11] Mihajlo A. Jovanovic, Fred S. Annexstein, and Kenneth A. Berman. Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella. University of Cincinnati, Laboratory for Networks and Applied Graph Theory, 2001. http://www.ececs.uc.edu/~mjovanov/Research/paper.ps.

[12] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. Technical report 99-1776. Cornell Computer Science, October 1999.

[13] Mojo Nation Technology Overview. 14 February 2000. http://www.mojonation.net/docs/technical_overview.shtml.

[14] Napster homepage, 2001. http://www.napster.com/.

[15] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Proc. ACM Symposium on Parallel Algorithms and Architectures*, June 1997.

[16] Sylvia Ratnasamy et al. A Scalable Content-Addressable Network. *Proceedings of the ACM SIGCOMM*, 2001.

[17] Sean Rhea et al. Maintenance-free Global Data Storage. *IEEE Internet Computing*, **5**(5), September/October 2001.

[18] Marco Schneider. Self-Stabilization. *ACM Computing Surveys*, **25**(1), March 1993.

[19] Kunwadee Sripanidkulchai. The popularity of Gnutella queries and its implications on scalability, February 2001. http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html.

[20] Michael Stonebraker et al. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, **5**(1), 1996.

[21] William Wong. Furi hompage, 2001. http://www.jps.net/williamw/furi/.

[22] Haruo Yokota, Yasuhiko Kanemasa, and Jun Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. *Proc. International Conference on Data Engineering (ICDE)*, 1999.