Athens University of Economics and Business

ELTRUN

The eBusiness Centre (www.eltrun.gr)

A Survey of
Peer-to-Peer File Sharing Technologies

White Paper

A Survey of Peer-to-Peer File Sharing Technologies

White Paper

Written by
Stephanos Androutsellis-Theotokis

ELTRUN,
Athens University of Economics and Business, Greece

Legal Notices

The information in this document is subject to change without notice. ELTRUN makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for its particular purpose. ELTRUN shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Table of Contents

Executive summary

In the recent years, the evolution of a new wave of innovative network architectures labeled *"peer-to-peer (p2p)"* has been witnessed. Such architectures and systems are characterized by direct access between peer computers, rather than through a centralized server. The recently formed *Peer-to-Peer Working Group*, a consortium including industry leaders aiming at the advancement of infrastructures and best-known practices for peer-to-peer computing, defines p2p as the "sharing of computer resources by direct exchange". Apart from resources, p2p offers a way of decentralizing administration (as well as cost).

File sharing is the dominant p2p application on the Internet, allowing users to easily contribute, search and obtain content.

Grid computing, which has emerged as a field distinguished from conventional computing by its focus on wide area distributed computing, large-scale resource sharing and problem solving is closely related to p2p. It is expected that there will be an even stronger convergence between them as p2p technologies become more sophisticated.

P2p file sharing architectures can be classified by their "degree of centralization", i.e. to what extent they rely to one or more servers to facilitate the interaction between peers. Three categories are identified: *Purely decentralized*, *partially centralized* and *hybrid decentralized*.

Furthermore, highly dynamic p2p networks of peers with complex topology can be differentiated by the degree to which they contain some structure or are created *ad-hoc*. By structure we refer to the way in which the content of the network is located: Is there a way of directly knowing which peers contain some specific content, or does one need to "randomly" search the entire network to locate it? Three categories of systems are examined: *Structured*, *loosely structured* and *unstructured*.

Various p2p architectures from these categories are examined with focus on the way they operate and how successfully they address issues such as scalability, network latency, security, privacy, anonymity and others. The shortcomings of these systems and the latest variations, developments and trends in p2p file sharing network design that aim at improving upon them, are discussed. Specific applications built on top of p2p infrastructures are beyond the scope of this report.

1. Introduction

In the recent years, the evolution of a new wave of innovative network architectures labeled *“peer-to-peer (p2p)”* has been witnessed. Such architectures and systems are characterized by direct access between peer computers, rather than through a centralized server. File sharing is the dominant p2p application on the Internet, allowing users to easily contribute, search and obtain content.

P2p file sharing architectures can be classified by their “degree of centralization”, i.e. to what extent they rely to one or more servers to facilitate the interaction between peers. Three categories are identified: *Purely decentralized*, *partially centralized* and *hybrid decentralized*.

Furthermore, highly dynamic p2p networks of peers with complex topology can be differentiated by the degree to which they contain some structure or are created *ad-hoc*. By structure we refer to the way in which the content of the network is located: Is there a way of directly knowing which peers contain some specific content, or does one need to “randomly” search the entire network to locate it? Three categories of systems are examined: *Structured*, *loosely structured* and *unstructured*.

Various p2p architectures from these categories are examined with focus on the way they operate and how successfully they address issues such as scalability, network latency, security, privacy, anonymity and others. The shortcomings of these systems and the latest variations, developments and trends in p2p file sharing network design that aim at improving upon them, are discussed. Specific applications built on top of p2p infrastructures are beyond the scope of this report.

1.1. What is p2p (and what isn't)

The new wave of innovative network architectures such as Napster, Gnutella, Seti@Home, Groove and many others has brought on a revolution in computing that, for lack of a better term, has been labeled peer-to-peer (p2p).

Several attempts have been made at identifying the defining features of p2p systems. In a nutshell, p2p is characterized by direct access between peer computers, rather than through a centralized server.

According to [33], p2p refers to applications that take advantage of resources (storage, cycles, content, human presence) available at the edges of the internet. The ``litmus test'' for p2p is:

- Does it treat variable connectivity and temporal network addresses as the norm?
  … and …
- Does it give the nodes at the edges of the network significant autonomy?

Another way to describe p2p systems, is to think about ownership: "Who owns the hardware that the services run on?". In this sense, p2p offers a way of decentralizing administration (as well as cost).

The recently formed Peer-to-Peer Working Group (www.p2pwg.org), a consortium including Hewlett-Packard, Intel, IBM and other industry leaders whose aim is to “facilitate and accelerate the advancement of infrastructure best-known practices for peer-to-peer computing” [34], defines p2p as the “sharing of computer resources by direct exchange”.

Another characteristic of (most) p2p systems is their self-organizing capacity: The topology of a p2p network must change as nodes (i.e. users, pc's) will enter or leave, in order to maintain its connectivity and performance.

File sharing remains the dominant (by far) p2p application on the internet [8], allowing users to easily contribute, search and obtain content. A term often used for this content is *Dark Matter*: "The universe is thought to consist primarily of undetected matter, called dark matter. The metaphor here is that the vast majority of computing resources are also not detectable or leveragable" – Lucas Gonze [35].

1.2. P2p and “the Grid”

Before proceeding to a more detailed description of p2p technology, let us consider for a moment its position with respect to “Grid” computing.

The fundamental concept of Grid Computing is “Enabling the coordinated use of geographically distributed resources, in the absence of central control, omniscience, strong trust relationship” [23].

Grid computing (or “the Grid”, as it is often referred to) has thus emerged as an important new field, distinguished from conventional computing by its focus on wide area distributed computing, large-scale resource sharing and problem solving in dynamic *virtual organizations* and, in some cases, high performance orientation [32].

Virtual organizations are sets of individuals and/or institutions defined by specific “sharing rules”. Some examples are: Application service providers; consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory; members of an industrial consortium bidding on a new aircraft; and members of a large, international, multi-layer high-energy physics collaboration [32].

The class of problems addressed by Grid computing in this context therefore includes authentication, authorization, resource discovery and access etc.

Peer to peer systems have much in common with Grid computing. However, the sharing that Grid computing is concerned with is not primarily file exchange, but rather direct access to computers, software, data and other resources [32].

One reason that Grid computing and peer-to-peer technologies have not overlapped significantly to date seems to be that p2p developers have focused mainly on vertically integrated solutions, rather than seeking to define common protocols that would allow for shared infrastructure and interoperability (a common practice for new market niches). Another is that the form of sharing targeted by p2p has been rather limited (e.g. file sharing with no access control).

However, as p2p technologies are becoming more sophisticated, it is expected that there will be a strong convergence between p2p and Grid computing [41].

As a first sign of such a convergence, the Global Grid Forum (GGF) [27], a community-initiated forum of individual researches and practitioners working on distributed computing, or Grid technologies and the Peer-to-Peer Working Group [28] have decided to unite their web sites. We read from the GGF's main page:

“The Global Grid Forum (GGF) is a community-driven set of working groups that are developing standards and best practices for distributed computing (‘Grids’ and ‘Meta-computing’) efforts, including those specifically aimed at very large data sets, high performance computing *but increasingly those efforts that industry is calling ‘Peer-to-peer'*.”

2. Classification of p2p file sharing architectures

In this Section we examine two general aspects of p2p architectures, according to which the p2p systems can be differentiated and categorized: The degree of centralization, and the network structure.

2.1. Degree of centralization

P2p file sharing architectures can be classified by their "degree of centralization", i.e. to what extent they rely to one or more servers to facilitate the interaction between peers. Three categories are identified:

- **Purely decentralized** p2p architectures (such as the original Gnutella architecture and Freenet).
  All nodes in the network perform exactly the same tasks, acting both as servers and clients, and there is no central coordination of their activities. The nodes of such networks are termed "servents" (SERVers+clieENTS).

- **Partially centralized** systems (such as Kazaa, Morpheus and more recently Gnutella).
  The basis is the same as with purely decentralized systems. However, some of the nodes assume a more "important" role than the rest of the nodes, acting as local central indexes for files shared by local peers. These nodes are called "Supernodes", and the way in which they are selected for these special tasks vary from system to system. It is important to note that these Supernodes do not constitute single points of failure for a p2p network, since they are dynamically assigned and in case they are subject to failure or malicious attack the network will take action to replace them with others.

- **Hybrid decentralized** architectures (such as Napster).
  There is a central server facilitating the interaction between peers by maintaining directories of the shared files stored on the respective PCs of registered users to the network, in the form of meta-data. The end-to-end interaction is between two peer clients, however these central servers facilitate this interaction by performing the lookups and identifying the nodes of the network (i.e. the computers) where the files are located. The terms "peer-through-peer" or "broker mediated" are sometimes used for such systems [23].

  Obviously in these architectures there is a single point of failure (the central server). This makes them vulnerable to censorship, technical failure or malicious attack, which in itself is enough to defeats the purpose of p2p as we view it.

  For the purposes of this survey, hybrid centralized systems are not considered real p2p systems, as they follow the standard client-server paradigm, and only the file transfer takes place between peers. We are therefore not going to examine hybrid centralized systems in extensive detail.

2.2. Network structure

P2p systems constitute highly dynamic networks of peers with complex topology. This topology creates an overlay network, which may be totally unrelated to the physical network that connects the different nodes (computers). P2p systems can be differentiated by the degree to which these overlay networks contain some structure or are created *ad-hoc*. By structure here we refer to the way in which the content of the network is located with respect to the network topology: Is there a way of directly knowing on which nodes some specific content is located, or do we need to "randomly" search the entire network to find it?

- In **unstructured** networks (such as Gnutella), the placement of data (files) is completely unrelated to the overlay topology. Since there is no information about which nodes are likely to have the relevant files, searching essentially amounts to random search, in which various nodes are probed and asked if they have any files matching the query.

  Unstructured p2p systems differ in the way in which they construct the overlay topology, and they way in which they distribute queries from node to node. The advantage of such systems is that they can easily accommodate a highly transient node population. The disadvantage is that it is hard to find the desired files without distributing queries widely. For this reason unstructured p2p systems are considered to be unscalable. However, as will be discussed in Section 4, work is done towards increasing the scalability of unstructured systems.

- **Structured** networks, (such as Chord, CAN, PAST, Tapestry etc.) have emerged mainly in an attempt to address the scalability issues that unstructured systems are faced with. The random search methods adopted by unstructured systems seem to be inherently unscalable [8], and structured systems were proposed, in which the overlay network topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. These systems provide a mapping between the file identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired file [8].

  Structured systems offer a scalable solution for exact-match queries, i.e. queries in which the complete identifier of the requested data object is known (as compared to keyword queries). There are ways to use exact exact-match queries as a substrate for keyword queries [14], however it is not clear how scalable these techniques will be in a distributed environment.

  The disadvantage of structured systems is that it is hard to maintain the structure required for routing in a very transient node population, in which nodes are joining and leaving at a high rate [8].

- **Loosely structured** networks (such as Freenet), are in between the two. File locations are affected by routing hints, but they are not completely specified, so not all searches succeed [8].

3. Overview of p2p file sharing architectures

The following Sections present a survey of current p2p file sharing architectures and systems, including their main characteristics and evolution. Each of the systems will be discussed separately but with focus on the way in which they complement each other and address each others shortcomings.

Table 1 illustrates the different categories of systems that will be examined. Note that structured and loosely structured systems are inherently purely decentralized.

	Unstructured Networks	Loosely Structured Networks	Structured Networks
Hybrid Decentralized	Napster		
Pure Decentralized	Gnutella	Freenet	Chord, Can, Tapestry
Partially Centralized	Kazaa, Gnutella		

Table 1. A classification of peer-to-peer file-sharing systems. Structured and loosely structured systems are inherently purely decentralized, while unstructured systems can be either pure or hybrid decentralized systems or partially centralized.

This survey does not describe all currently designed and deployed systems, however representative systems from each category are discussed, and the points made are valid for all similar p2p architectures. The latest advances and trends in these systems will be discussed in Section 4.

3.1. Unstructured systems

3.1.1. Hybrid decentralized unstructured systems

3.1.1.1. Napster

As mentioned earlier in this report, we do not really consider hybrid decentralized systems to be real p2p systems. For completeness we briefly outline the principle behind Napster.

Figure 1 illustrates the architecture of Napster.

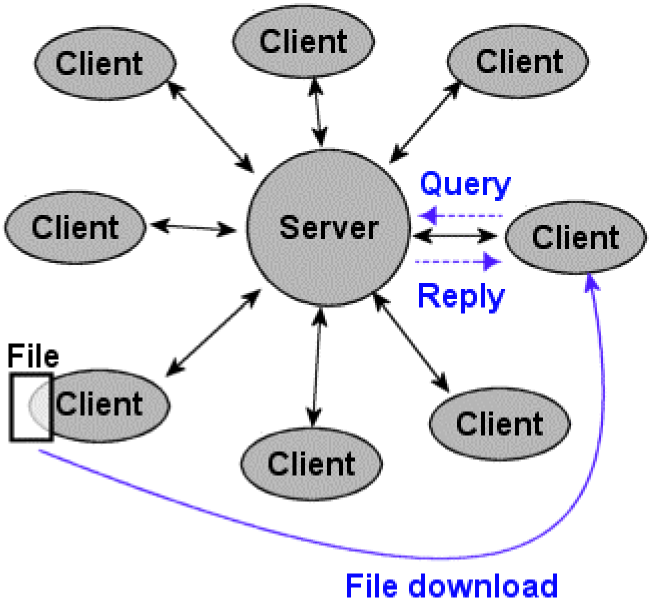

Figure 1. Architecture of Napster. A central directory server maintains an index of the metadata for all files in the network.

The systems comprises a network of registered users running some client software, and a central directory server maintaining:

- An index with metadata (file name, time of creation etc.) of all the files in the network.
- A table of registered user connection information (IP addresses, connection speeds etc.)
- A table listing the files that each user holds and shares in the network.

On startup, the client contacts the central server and reports a list with the files it maintains.

When the server receives a query from a user, it searches for matches in its index, returning a list of users that hold the matching file. The user then opens a direct connection with the peer that holds the requested file, and downloads it (see Figure 1).

The advantage of Napster and similar systems is that they are simple and they locate files quickly and efficiently. The main disadvantage of course is that such centralized systems are vulnerable to censorship, malicious attack and technical failure. Furthermore, these systems are inherently not largely scalable, as there are bound to be limitations to the size of the server database and its capacity to respond to queried.

From another standpoint, these systems do not offer an acceptable p2p solution, since the content shared, or at least descriptions of it and the ability to access it are controlled by the single institution, company or user maintaining the central server.

3.1.2. Purely decentralized unstructured systems

3.1.2.1. Gnutella (original architecture)

The Gnutella network, which originated as a project at Nullsoft, a subsidiary of America on Line, is one of the most interesting p2p architectures to study due to its open architecture, achieved scale and self-organizing structure [18].

Like most p2p systems, Gnutella builds, at the application level, a virtual overlay network with its own routing mechanisms [5]. It therefore works as a distributed file storage system, allowing its users to specify directories on their machines which they want to share with other peers. Figure 2 taken from [22] without permission by the author, shows a "snapshot" of the Gnutella network taken in January 2000.

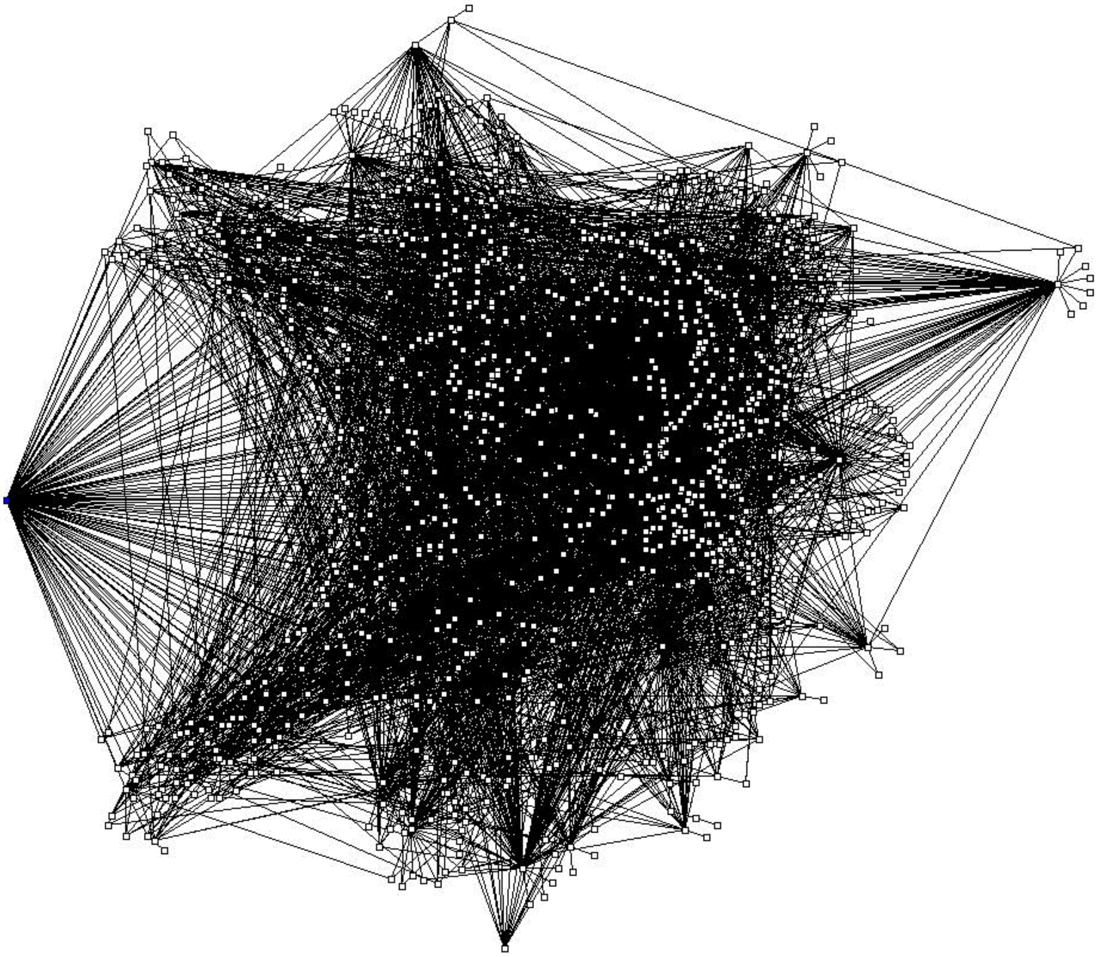

Figure 2. A snapshot of the Gnutella network on January 27 2000 (from [22]).

Since it is a purely decentralized architecture there is no central coordination of the activities in the network and users connect to each other directly through a software application which functions both as a client and a server, and is therefore referred to as a *servent*. An instance of this software running on a particular machine is also referred to as a *host*.

Gnutella uses IP as its underlying network service, while the communication between hosts is specified in a form of application level protocol supporting four types of messages [11]:

- **Ping**
  A request for a certain host to announce itself.
- **Pong**
  Reply to a Ping message. It contains the IP and port of the responding host and number and size of files shared.
- **Query**
  A search request. It contains a search string and the minimum speed requirements of the responding host.
- **Query hits**
  Reply to a Query message. It contains the IP and port and speed of the responding host, the number of matching files found and their indexed result set.

After joining the Gnutella network (by using hosts such as gnutellahosts.com), a node sends out a Ping message to any node it is connected to. The nodes send back a pong message identifying themselves, and also propagate the ping message to their neighbors.

In order to locate a file, in unstructured systems such as gnutella, random searches are the only option since the nodes have no way of guessing where (in which hosts) the files may lie.

Gnutella originally uses TTL-limited flooding (or broadcast) to distribute Ping and Query messages: Each Gnutella host forwards the received messages to all of its neighbors. The response messages are routed back along the opposite path through which the original request arrived. To limit the spread of messages through the network, each message header contains a time-to-live (TTL) field. At each hop the value of this field is decremented, and when it reaches zero the message is dropped.

The above is implemented by flagging each message with a unique identifier, and by equipping each host with a dynamic routing table of message identifiers. Since the response messages contain the same ID as the original messages, the host checks its routing table to determine along which link the response message should be forwarded. In order to avoid loops, the nodes use the unique message identifiers to detect and drop duplicate messages. This technique improves efficiency and preserves network bandwidth [22].

Once a node receives a QueryHit message, indicating that the target file has been identified at a certain node, it initiates a direct out-of-network download, establishing a direct connection between the source and target node.

Figure 3 illustrates an example of the Gnutella search mechanism [36].

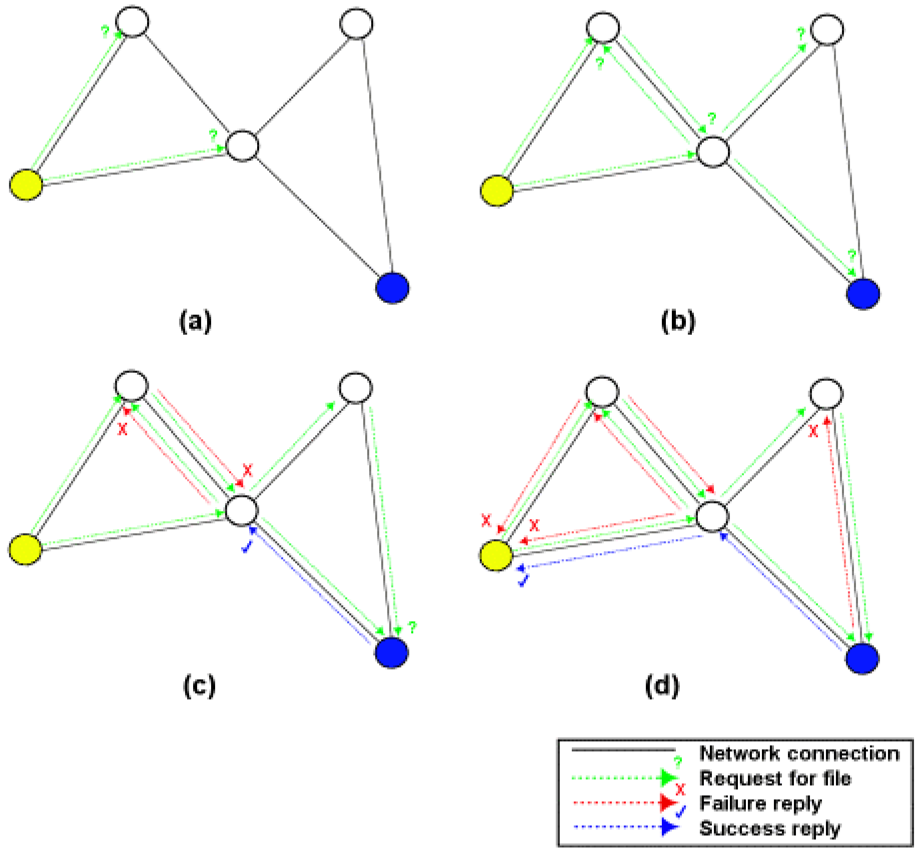

Figure 3. An example of the Gnutella search mechanism.

One disadvantage of Gnutella and other similar systems is that the TTL effectively segments the Gnutella network into subnets, imposing on each user a virtual “horizon” beyond which their messages cannot reach [22]. If on the other hand the TTL is removed, the network would be swamped with requests. This suggests that the Gnutella network is faced with a scalability problem.

In [16], several alternative query distribution methods are investigated, and the use of multiple parallel random walks (each node chooses a neighbor at random and send the query only to it) is proposed instead of flooding.

In [15], algorithms for proactive replication of objects (files) in the network are studied. Passive replication of files takes place as users transfer them from other nodes to their own. In these proactive replication studies, it was found that the optimal replication scheme for random search is to replicate objects proportionally to the square root of their query rate.

These two approaches combined were found in [16] to significantly improve the performance of the system as measured by query resolution time (in terms of numbers of hops), per-node query load and message traffic generated.

Another disadvantage of Gnutella is that it generates a considerable amount of query traffic on the networks. This problem is alleviated by using parallel random walks as above, and by recent work discussed in Section 4.

3.1.3. Partially centralized unstructured systems

3.1.3.1. Kazaa, Morpheus

Kazaa and Morpheus are two similar partially centralized systems which use the concept of "SuperNodes", i.e. nodes that are dynamically assigned the task of servicing a small subpart of the peer network by indexing and caching files contained in the part of the network they are assigned to. Both Kazaa and Morpheus are proprietary and there is no detailed documentation on how they operate.

Peers are automatically elected to become SuperNodes if they have sufficient bandwidth and processing power (although a configuration parameter allow users to disable this feature), using proprietary algorithms licensed from FastTrack [44].

In Morpheus a central server provides new peers with a list of one or more SuperNodes with which they can connect. SuperNodes index the files shared by peers connected to them, and proxy search requests on behalf of these peers [24]. Queries are therefore sent to superNodes, not to other peers.

The advantage of partially centralized systems is that discovery time is reduced in comparison with purely decentralized systems, while there is still no unique point of failure such as one single central server. If one or more SuperNodes go down, the nodes connected to them can open new connections with other SuperNodes, and the network will continue to operate. In the event that a very large number or even all SuperNodes go down, the existing peers can become SuperNodes themselves.

3.1.3.2. Gnutella (more recent architecture)

The concept of SuperNodes has also been proposed in a more recent version of the Gnutella protocol. A mechanism for dynamically selecting SuperNodes organizes the Gnutella network into an interconnection of SuperPeers (as they are referred to) and client nodes [26].

As a node with enough CPU power joins the network, it immediately becomes a SuperPeer and establishes connections with other SuperPeers, forming a flat unstructured network of SuperPeers. It also sets the number of clients required for it to remain a SuperPeer. If it receives at least the required number of connections to client nodes within a specified time, it remains a SuperPeer. Otherwise it turns into a regular client node. If no other SuperPeer is available, it tries to become a SuperPeer again for another probation period.

3.2. Loosely structured systems

3.2.1. Freenet

Freenet [12] is a purely decentralized loosely structured system, operating as a self-organizing p2p network. It essentially pools unused disk space in peer computers to create a collaborative virtual file system providing both security and
publisher anonymity.

In this respect, a main difference from other systems such as Gnutella is that Freenet provides file-storage service, rather than file-sharing service. Whereas in Gnutella files are only copied to other nodes when these nodes request them, In Freenet files are pushed to other nodes for storage, replication and persistence [20].

Furthermore, Freenet makes it infeasible to discover the true origin or destination of a file passing through the network, and difficult for a node operator to determine (or be held responsible for) the actual physical contents of their own node.

Freenet nodes maintain their own local datastore, which they make available to the network for reading and writing, as well as a dynamic routing table containing addresses of other nodes and the *keys* (file identifiers) they are thought to hold.

Files in Freenet are identified by binary keys. There are three types of keys: keyword-signed keys, signed-subspace keys and content-hash keys. To search for a file, the user sends a request message specifying the key and a timeout (hops-to-live) value.

Messages in Freenet always include an ID (for loop detection), a hops-to-live value, source and destination, and are of the following types:

- **Data request**. Additional field: Key.
- **Data reply**. Additional field: Data.
- **Data failed**. Additional fields: Location and reason.
- **Data insert**. Additional fields: Key and data.

Each Freenet node maintains a common stack storing:

- **ID**: File identifier
- **Next hop**: Another node that stores this ID.
- **File**: The file identified by the id, stored on the local node.

Joining the Freenet network is simply a matter of first discovering the address of one or more existing nodes, and then starting to send messages.

In order to insert new files to the network, the user must first calculate a binary file key for it. She then sends an insert message to her own node specifying the proposed key and a hop-to-live value (this will determine the number of nodes to store it on). When a node receives the insert message, it first checks to see if the key is already taken. If the key is found to be taken, the node returns the pre-existing file as if a request were made for it. If the key is not found, the node looks up the nearest key in its routing table, and forwards the insert message to the corresponding node. If the hops-to-live limit is reached without any key collision, an “all clear” result will be propagated back to the original inserter, informing that the insert was successful.

By this mechanism, newly inserted files are placed on nodes possessing files with similar keys. Furthermore, new nodes can use inserts to announce their presence to the rest of the network. Finally, malicious attempts to supplant existing files by inserting junk will end up in the existing files being spread further.

Instead of broadcasting requests for files to all neighbors, as Gnutella does, Freenet uses a “chain-mode” file discovery mechanism. The basic model is that requests for keys are passed along from node to node through a chain of requests in which each node makes a local decision about where to send the request next.

If a node receives a request for a file that it stores locally, the search stops and the data is forwarded back to the requestor. If the node does not store the file that the requestor is looking for, it forwards the request to one of its neighbors that is more likely to have the file, by searching for the "closest" ID in the stack. The messages therefore form a chain, as they propagate from node to node. In order to avoid huge chains, messages time out after passing through a certain number of nodes, based on the hops-to-live value they carry. Nodes also store the ID and other information of the requests they have seen, in order to handle data reply messages and request failed messages.

If a node receives a backtracking request failed message from its downstream node, then it selects the "next best" node from its routing stack and forwards the request to it. If all nodes in the routing table have been explored in this way and failed, it sends back a request failed message to the node from which the originally data request was received.

If the file is eventually found at a certain node, a reply is passed back through each node that forwarded the request to the original node that started the chain. This data reply message will include the actual data, which will be cached in all intermediate nodes for future requests. A subsequent request to the same key will be served immediately with the cached data. A request to a "similar" key (determined by lexicographic distance) will be forwarded to the node that provided the data previously.

In this way there is no direct connection between requestor and actual data source, anonymity is maintained, and the owners of files cached cannot be held responsible for the content of their caches (file encryption with original text names as key is a further measure that is taken). Figure 4 illustrates the chain mode file discovery mechanism.

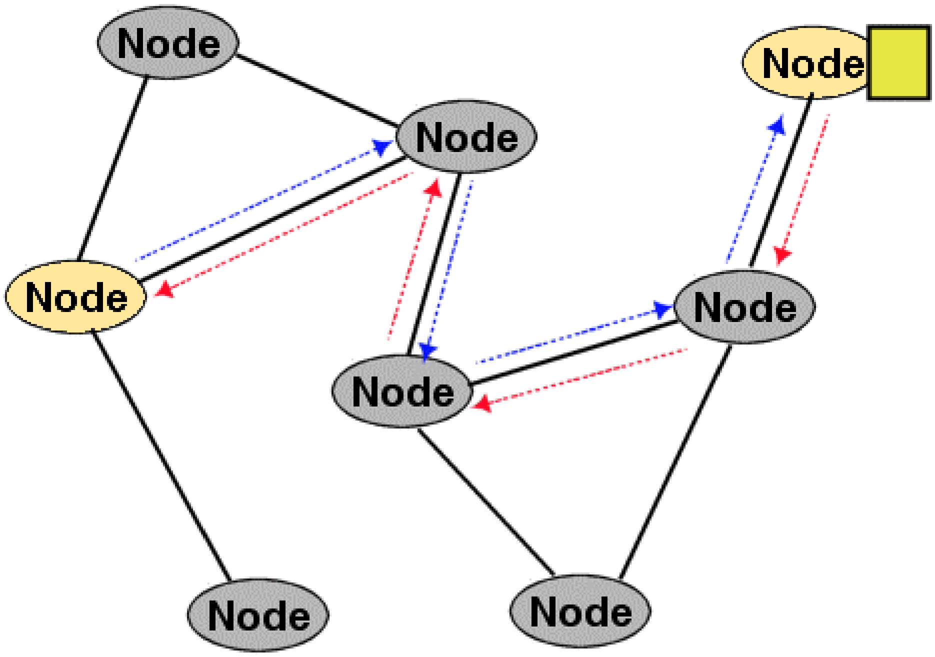

Figure 4. The Freenet chain mode file discovery mechanism. The query is forwarded from node to node using the routing table, until it reaches the node which has the requested data. The reply is passed back to the original node following the reverse path.

To keep the actual data source anonymous, two further measures are taken in Freenet:

- Any node along the reply path can change the reply message and claim to be the source of the data. Since the data will be cached on this node as well, it will actually be able to serve future requests to the same data.

- The hops-to-live counter is randomly initiated in order to obscure the distance from the originator.

The Freenet network is described as “loosely structured” because, based on the information in its stack, a node can produce an estimate of which node is more likely to contain the requested data. In this way, instead of blindly broadcasting data request messages to all neighbors, the chain mode is used to propagate a request through a path that is more likely to succeed in the data being found.

In order to address the problem of finding the keys that correspond to a specific file, Freenet suggests a special class of lightweight files called “indirect files”. When a real file is inserted, the author could also insert a number of indirect files each containing a pointer to the real file, named according to search keywords chosen by her. These indirect files would differ from normal files in that multiple files with the same key (i.e. search keyword) would be permitted to exist, and requests for such keys would keep going until a specified number of results were accumulated, instead of stopping at the first file found. The problem of managing the large volume of such indirect files remains open.

Following are some further features resulting from the properties of the Freenet network [25]:

- Nodes tend to specialize in searching for similar keys over time, as they get queries from other nodes for similar keys.

- Nodes store similar keys over time, due to the caching of files as a result of successful queries.

- Similarity of keys does not reflect similarity of files.

- Routing does not reflect underlying network topology.

3.3. Structured systems

3.3.1. Chord

Chord [17] provides support for just one operation: given a key, it maps the key onto a node. Data location can then be implemented on top of Chord by associating a key with each data item and storing the key/data item pair at the node to which the key maps.

Each Chord node needs routing information for only a few other nodes.(only *O(log N)* for an *N*-node system in the steady state), and resolves all lookups via *O(log N)* messages to other nodes. Performance degrades gracefully when routing information becomes out of date due to nodes joining and leaving the system; only one piece of information per node need be correct in order for Chord to guarantee correct (though slow) routing of queries.

In Chord, unique IDs are associated with both data items and nodes by means of a variant of consistent hashing. Consistent hashing [13] tends to balance load, since each node receives roughly the same number of keys and involves relatively little movement of keys when nodes join or leave the network.

As nodes enter the network, they are assigned unique IDs by hashing their IP address.

Keys (file ids) are assigned to nodes as follows. Identifiers are ordered in an "identifier circle" modulo 2^m (Figure 5 shows an identifier circle with *m=3*). Key *k* is assigned to the first node whose identifier is equal to or follows (the identifier of) *k* in the identifier space. This node is called the successor node of key *k*.

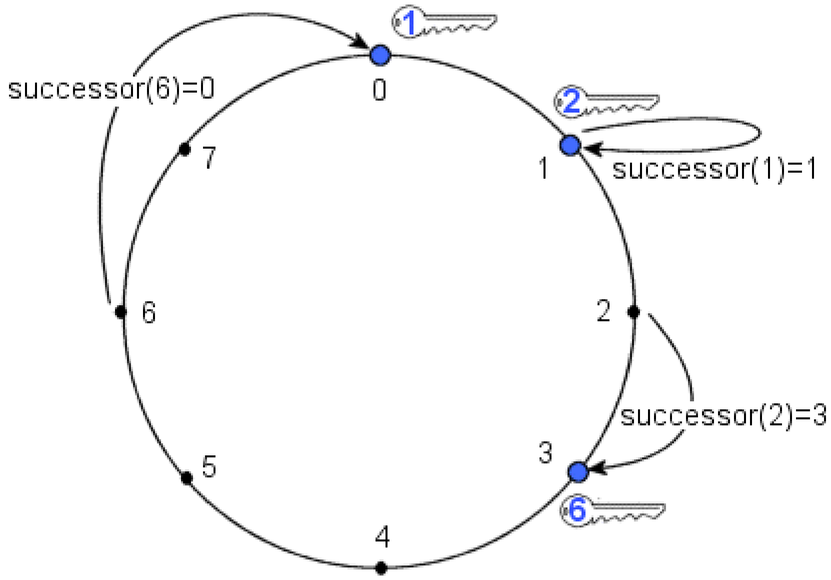

Figure 5. A Chord identifier circle consisting of the three nodes 0,1 and 3. In this example, key 1 is located at node 1, key 2 at node 3 and key 6 at node 0.

As a result, when a node *n* joins the network, certain keys previously assigned to *n*'s successor will become assigned to *n*. When node *n* leaves the network, all keys assigned to it will be reassigned to its successor. These are the only changes in key assignments that need to take place in order to maintain load balance.

The only routing information required is for each node to be aware of its successor node on the circle. Queries for a given identifier are passed around the circle via these successor pointers until they first encounter a node that succeeds the identifier. This is the node the query maps to.

The resolution scheme described above is inefficient, since it may require traversing all *N* nodes to find the appropriate mapping. In order to accelerate the process, Chord maintains additional routing information. This additional information is called a "finger table", in which each entry *i* points to the successor of node *n+2i* (see Figure 6). In order for a node *n* to perform a lookup for key *k*, the finger table is consulted to identify the highest node *n'* whose id is between *n* and *k*. If such a node exists, the lookup is repeated starting from *n'*. Otherwise, the successor of *n* is returned. Using the finger table, lookups can be completed in time *log(N)*.

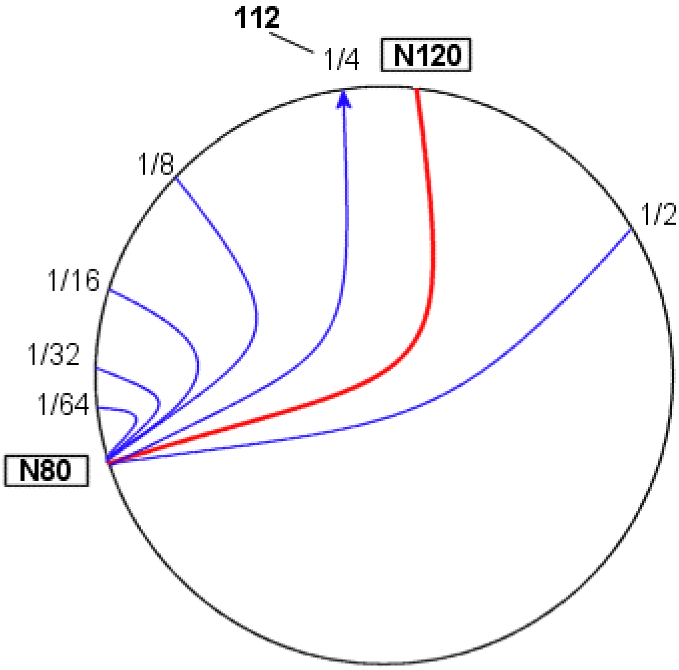

Figure 6. The Chord finger table. Finger i points to Successor of n+2i.

Achord [2] is proposed as a censorship resistant variant of Chord, by limiting each nodes knowledge of the network in ways similar to Freenet (Section 3.2.1). For example instead of using the *find_successor* method which returns a pointer to a node's successor, Achord uses *connect_to_successor*, which performs a tunneled connection path to the node, without however returning the node id. In this way the identity of the nodes responsible for storing a certain key is protected.

3.3.2. CAN

CAN (Content Addressable Network) [21] is essentially a distributed, internet-scale hash table that maps file names (whether well known or discovered through some external mechanism) to their location in the network.

The basic operations performed by CAN are the insertion, lookup and deletion of *(key, value)* pairs in the distributed hash table. Each CAN node stores a part (called a “zone”) of the hash table, as well as information about a small number of “adjacent” zones in the table. Requests to insert, lookup or delete for a particular key are routed via intermediate nodes to the node that maintains the zone containing the key.

The main features of CAN are:

- Purely decentralized.
- Scalable (nodes maintain only a small part of the state of the system, independent of the number of nodes in the system).
- Fault-tolerant (nodes can route around failures).

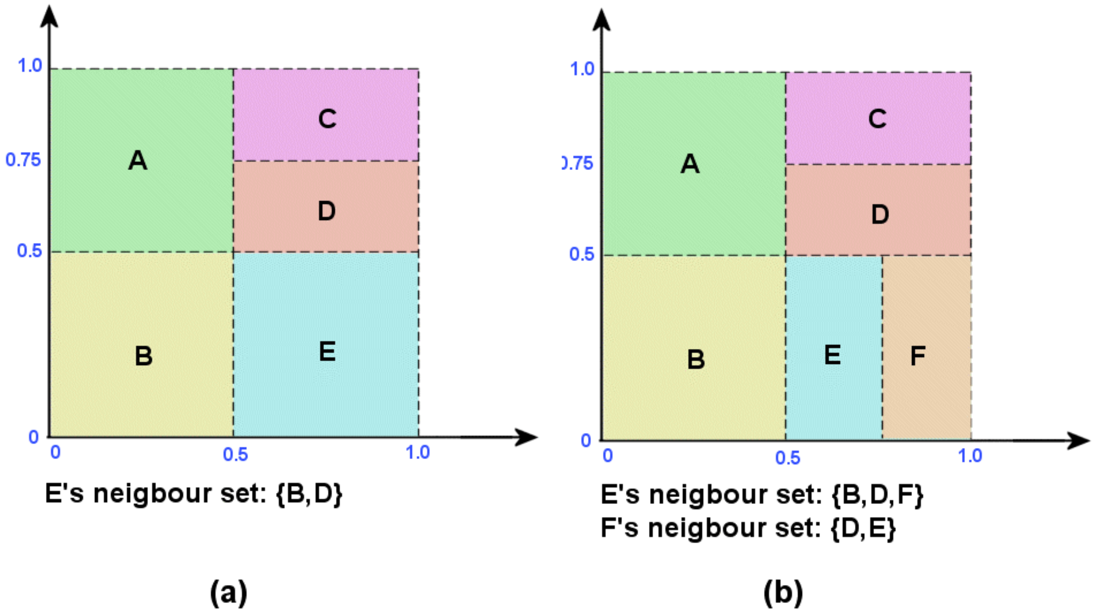

Figure 7. CAN: (a) Example 2-d [0,1]X[0,1] coordinate space partitioned between 5 CAN nodes; (b) Example 2-d space after node F joins.

CAN uses a virtual *d*-dimensional Cartesian coordinate space (see Figure 7 to store *(key K, value V)* pairs as follows: First, *K* is deterministically mapped onto a point *P* in the coordinate space. The *(K, V)* pair is then stored at the node that owns the zone within which point *P* lies.

To retrieve the entry corresponding to *K*, any node can apply the same deterministic function to map *K* to *P* and then retrieve the corresponding value *V* from *P*. If *P* is not owned by the requesting node, the request must be routed from node to node until it reaches the node in whose zone *P* lies.

CAN nodes learn and maintain the IP addresses of nodes that hold coordinate nodes adjoining their own in a routing table that enables routing between arbitrary points in space. Intuitively routing in CAN works by following the straight line path through the Cartesian space from source to destination coordinates.

New nodes that join the CAN system are allocated their own portion of the coordinate space by splitting the allocated zone of an existing node in half, as follows:

- The new node identifies a node already existing in CAN, using some bootrstrap mechanism as in [42].
- Using the CAN routing mechanism, it randomly chooses a point *P* in the space and sends a JOIN request to the node whose zone contains *P*. The zone will be split, and half will be assigned to the new node.
- The new node learns the IP addresses of its neighbors, and the neighbors of the split zone are notified so that routing can include the new node.

When nodes leave CAN, the zones they occupy and the associated hash table entries are explicitly handed over to one of their neighbors.

Under normal conditions a node sends periodic update messages to each of its neighbors giving its zone coordinates, list of neighbors and their zone coordinates. If there is prolonged absence of such an update message, the neighbor nodes realize there has been a failure, and initiate a controlled takeover mechanism. If many of the failed nodes neighbors also fail, an expanding ring search mechanism is initiated by one of the neighboring nodes, to identify any functioning nodes outside the failure region.

Following are a list of design improvements performed over the basic design described above:

- Use of multi-dimensional coordinate space for improving network latency and fault tolerance with a small routing table size overhead.

- Use of multiple coordinate spaces (realities) for fault tolerance.

- Better routing metrics, by taking into account the underlying IP topology and connection latency alongside the Cartesian distance between source and destination.

- Overloading coordinate zones by allowing multiple nodes to share the same zone for improved fault tolerance, reduced per-hop latency and path length.

- Use of multiple hash functions to map the same key onto different points in the coordinate space for replication.

- Topologically-sensitive network construction, assuming the existence of a set of machines that act as landmarks on the internet.

- More uniform partitioning when a new node joins by preferring to split larger zones.

- Use of caching and replication techniques.

3.3.3. Tapestry

Tapestry [19] is a self-administering, fault-tolerant location and routing infrastructure that provides routing of messages directly to the “closest” copy of an object (or service) using only point-to-point links between nodes and without centralized resources. Tapestry is a fundamental component of the OceanStore persistent storage system [37], [38].

Tapestry employs randomness to achieve both load distribution and routing locality, and has it roots in the Plaxton distributed search technique [39] described below. An architecture is thus proposed for creating an environment that offers system-wide stability, transparently masking faulty components, bypassing failed routes, removing nodes under attack from service and rapidly adapting communication topologies to circumstances.

Location information is distributed within the routing infrastructure and is used for incrementally forwarding messages from point to point until they reach their destination. This information is repairable “soft-state”, it's consistency is checked on the fly, and if lost due to failures or destroyed, it is easily rebuilt or refreshed.

The topology of the location and routing infrastructure is self-organizing as routers, nodes and data repositories will come and go and network latencies will vary as individual links will fail or vary their rates.

As mentioned above, the inspiration for Tapestry's design are the location and routing mechanisms introduced by Plaxton, Rajamaran and Richa [39], in which they present a distributed datastructure optimized to support a network overlay for locating named objects and routing messages to those objects. This datastructure, also called a *Plaxton mesh*, allows nodes to locate objects and route messages to them across an arbitrary-sized network while using a small, constant-sized routing map at each hop.

Note that the Plaxton mesh is assumed to be a static data structure, without node insertions and deletions, which is not of course the case in p2p networks.

The nodes of the Plaxton mesh can take on the role of servers (where objects are stored), routers (which forward messages) and clients (origins of requests).

A local routing map, or *neighbor map* is used at each node, to incrementally route messages to the node with the destination ID digit by digit, from the right to the left. For example Figure 8 illustrates the path taken by a message from node *67493* to node *34567* in a Plaxton mesh using decimal digits of length 5. The digits are resolved right to left as

$$xxxx7 \to xxx67 \to xx567 \to x4567 \to 34567$$

where *x*'s represent wildcards.

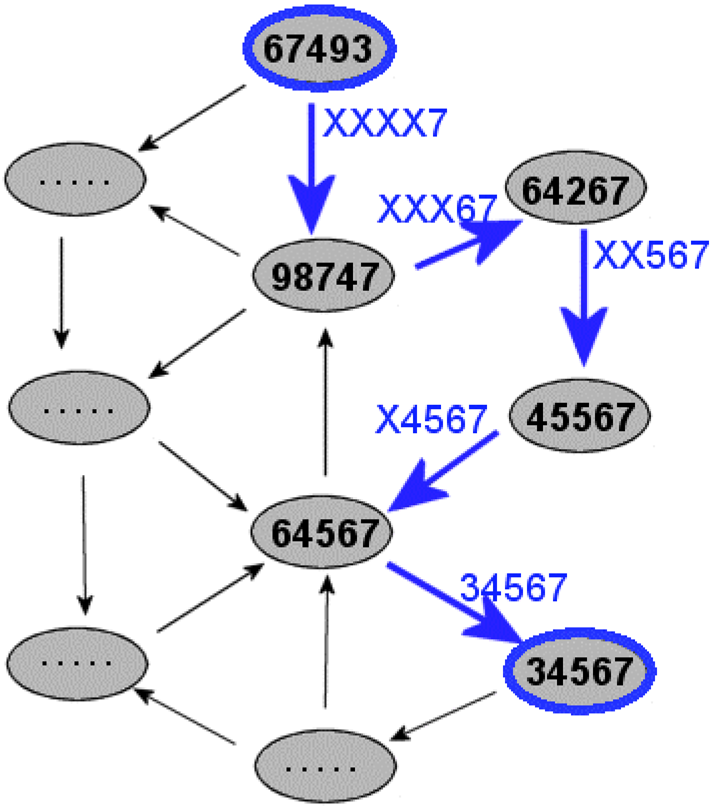

Figure 8. Tapestry: Plaxton routing example. Shows the path taken by a message originating from node 67493 destined for node 34567 in a Plaxton mesh using decimal digits of length 5.

Each node *N* therefore has a neighbor map with multiple levels, where each level represents a matching suffix up to a digit position in the ID.

For example the *5th* entry for the *3rd* level for node *67493* is the node closest to *67493* in network distance which ends in *..492*. Table 2 shows the neighbor map held by this node.

07493	x0493	xx093	xxx03	xxxx0
17493	x1493	xx193	xxx13	xxxx1
27493	x2493	xx293	xxx23	xxxx2
37493	x3493	xx393	xxx33	xxxx3
47493	x4493	xx493	xxx43	xxxx4
57493	x5493	xx593	xxx53	xxxx5
67493	x6493	xx693	xxx63	xxxx6
77493	x7493	xx793	xxx73	xxxx7
87493	x8493	xx893	xxx83	xxxx8
97493	x9493	xx993	xxx93	xxxx9

Table 2. The neighbor map held by Tapestry node with ID 67493. Each entry in this table corresponds to a pointer to another node.

The result is a neighbor map of constant size. In a system with *N*-sized namespace using ids of base *b*:

$$NeighborMapSize = entriesPerMap \text{ X } NoOfMaps = b \; Log_b(N)$$

A server *S* publishes that it has an object *O* by routing a message to a unique node *R* that is decided to be the "root" node for object *O*, storing along the way information in the form of a mapping *(object id O, server id S)*.

During a location query, messages destined for *O* are initially routed towards *O*'s root until a node is encountered containing the location mapping for *O*.

The root node therefore serves the important role of providing a guaranteed node where the location for that object can be found. Plaxton uses a globally consistent deterministic algorithm for choosing root nodes.

Overall, the benefits and limitations of the Plaxton mesh are the following:

- Benefits:
    - Simple fault handling, due to the potential to route around a single link or node, by choosing a node with a similar suffix.
    - Scalability, with the only bottleneck existing at the root nodes.
    - Exploitation of locality due to the routing process.
    - Proportional route distance.
- Limitations:
    - Global knowledge required for identifying root nodes.
    - Root node vulnerability.
    - Lack of ability to adapt.

Tapestry, based on the Plaxton mesh, extends its design to provide adaptability, fault tolerance as well as several optimizations, as described in [19] and outlined below:

- Each node additionally maintains a list of back-pointers, which point to nodes here it is referred to as a neighbor. These are used in the node dynamic node insertion algorithms to generate the appropriate neighbor maps for new nodes.

- The concept of distance between nodes becomes semantically more flexible, and locations of more than one replica of an object are stored, allowing the application architecture to define how the "closest" node will be interpreted. For example, in the Oceanstore architecture [37], a "freshness" metric is incorporated in the concept of distance, which is taken into account when finding the closest replica of a document.

- The soft-state or announce/listen approach [40] is adopted by Tapestry to detect, circumvent and recover from failures in routing or object location. Additionally, the neighbor map is extended to maintain two backup neighbors in addition to the closest/primary neighbor. Furthermore, to avoid costly reinsertions of nodes after failures, when a node realizes that a neighbor is unreachable, instead of removing its pointer it temporarily marks it as invalid in the hope that the failure will be repaired and in the meantime routes messages through alternative paths.

- To avoid the simple point of failure that root nodes constitute, Tapestry assigns multiple roots to each object through a globally constant hashing algorithm on the node IDs. This enables a tradeoff between reliability and redundancy.

- A distributed algorithm called *surrogate routing* is employed to compute a unique root node for an object in a globally consistent fashion, given the non-static set of nodes in the network

- Dynamic algorithms are employed for node insertion, populating neighbor maps and notifying neighbors of new node insertions.

- A set of optimizations improve performance by adapting to environment changes. Tapestry nodes tune their neighbor pointers by running refresher threads that update network latency values between their neighbors. Algorithms are implemented to detect query hotspots and offer suggestions as to where additional copies of objects can be placed to significantly improve query response time. A “hotspot cache” is also maintained at each node.

4. Shortcomings and improvements of p2p systems

This Section discusses recent work done in order to improve the performance of both structured and unstructured p2p file sharing systems.

4.1. Unstructured p2p systems

The main characteristic of unstructured p2p systems is that the placement of data is completely unrelated to the overlay topology. As was discussed in Section 2, since there is no information about which nodes might be likely to have the relevant files, file location on such networks essentially amounts to random search.

Such random search methods seem to be inherently unscalable [8] and as a result a number of structured p2p systems were proposed, as discussed in Section 3.3. In these systems the topology is tightly controlled and files are placed at precisely specified location. By the use of distributed routing tables, queries can be efficiently routed to the node with the desired file. These systems thus offer a scalable solution for “exact-match” queries (as opposed to keyword queries) [8].

According to [8], if scalability concerns were removed from unstructured p2p systems, they might be the preferred choice for file-sharing and other applications where the following assumptions hold:

- Keyword searching is the common operation.
- Most content is typically replicated at a fair fraction of participating sites.
- The node population is highly transient.

It therefore seems worthwhile to attempt to improve the scalability of unstructured systems. Work in this direction is being done by several research groups.

Several improvements to the basic Gnutella search method have been proposed.

In [8] an algorithm is proposed that uses the heterogeneity of unstructured systems (and specifically Gnutella) to improve their scalability and search efficiency.

In [16], the use of multiple parallel random walks is proposed instead of the flooding mechanism described in Section 3.1.2.1.

In [15], the use of proactive replication schemes, in which files may be replicated at nodes even though the nodes have not requested them, are found to improve the performance of the system.

In [8], a distributed flow control and topology creation algorithm is proposed which:

- Restricts the flow of queries into each node so that they won't become overloaded and
- Dynamically evolves the overlay topology so that queries flow towards nodes that have sufficient capacity to handle them.

In this approach, the “capacity” of a node is assumed to denote the maximum number of messages that the node is willing/able to process over a given time. Periodically, nodes check whether they are overloaded, i.e. whether their total incoming query rate exceeds their capacity. If this is the case, they attempt to adapt their topology by disconnecting from neighbors with high incoming query rates, and redirecting them to other nodes with higher capacities. Intuitively, links are set up between nodes with high query rates and nodes with high capacities, getting the overloaded nodes out of the way.

Furthermore, each node maintains information about the messages recently exchanged with its neighbors, as well as the capacities and incoming message rates of its neighbors. When performing random walks to search for files, the queries are directed from a node to the neighbor that has the more spare capacity for receiving queries from this node. This design therefore exploits and takes advantage of the heterogeneity of p2p networks.

Another approach that takes advantage of the different connectivity and forwarding capacities of nodes in p2p systems is proposed in [30].

In [5], the connectivity and reliability of unstructured p2p networks (and in particular Gnutella) is studied. p2p networks such as Gnutella exhibit the properties of so called *power-law networks*, in which the number of nodes with L links is proportional to L^{-k}, where k is a network dependent constant. In other words, most nodes have few links, thus a large fraction of them can be taken away without seriously damaging the network connectivity, while there are a few highly connected nodes which, if taken away, will cause the whole network to be broken down in pieces.

One implication of this is that such networks are robust when facing random node attacks, however vulnerable to well-planned attacks.

Measurements of the Gnutella network and the traffic it generates [5] showed that in 2000, 95\% of any two nodes where less than 7 hops away, the average traffic was 6Kbps per connection and the total traffic (excluding file transfers) was estimated at 330TB/month, which amounted to about 1.7\% of the total traffic in the US internet backbone in December 2000 (as reported in [11]).

The topology mismatch between the Gnutella network and the underlying physical network infrastructure was also documented in [5].

4.2. Structured p2p systems

It is argued in [7] that by creating keys for accessing data items (i.e. “virtualizing” the names of the data items) two main problems arise:

- Locality is destroyed.
  Data items (i.e. files) from a single site are not usually co-located, meaning that opportunities for enhanced browsing, pre-fetching and efficient searching are lost.
- Useful application level information is lost.
  The data used by many applications is naturally described using hierarchies, which expose relationships between items near to each other. The virtualization of the file namespace by generating keys discards this information.

A more sophisticated approach which allows systems to exploit locality between objects is proposed in [7].

The resilience of structured p2p systems in the face of a very transient user population and the difficulty of maintaining the structure required for routing to function efficiently when nodes are joining and leaving at a high rate, are considered in [8]

Furthermore, while structured systems are designed for exact-match queries, it is not yet demonstrated that a full range of partial query techniques can be scalably implemented [8].

Bibliography

[43] DJ. Brailer. Connection tops collection. Health Management Technology, August 2001.

[31] Y.Chen, RH. Katz, and JD. Kubiatowicz. Scan: A dynamic, scalable and efficient content distribution network. In Proceedings of International Conference on Pervasive Computing, 2000.

[20] I.Clake, TW. Hong, O.Sanberg, and B.Wiley. Protecting free expression online with freenet. IEEE Internet Computing, 6(1):40--49, January-February 2002.

[12] I.Clarke, O.Sandberg, and B.Wiley. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, June 2000.

[15] E.Cohen and S.Shenker. Optimal replication in random search networks. Preprint, Optional 2001.

[3] BF. Cooper and H.Garcia-Molina. Peer-to-peer resource trading in a reliable distributed system. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[40] S.Deering. Host extensions for ip multicasting. Technical Report RFC-1112, IETF, SRI International, Menlo Park, CA, August 1998.

[44] The FastTrack website: http://www.fasttrack.nu.

[41] I.Foster. Internet computing and the emerging grid. Nature Web Matters, 2000.

[36] I.Foster. Large scale networked systems. Lecture Notes, 2002.

[32] I.Foster, C.Kesselman, and S.Tuecke. The anatomy of the grid. Intl. J. Super-computer Applications, 2001.

[42] P.Francis. Yoid: Extending the internt multicast architecture. Unpublished Paper, April 2000.

[9] MJ. Freedman, E.Sit, J.Cates, and R.Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[18] The gnutella homepage: http://gnutella.wego.com.

[27] The Global Grid Forum website: http://www.gridforum.org.

[1] S.Hand and T.Roscoe. Mnemosyne: Peer-to-peer steganographic storage. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[2] S.Hazel and B.Wiley. Achord: A variant of the chord lookup service for use in censorship resistant peer-to-peerpublishing systems. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[7] P.Heleher, B.Bhattacharjee, and B.Silaghi. Are vitrualized overlay networks too much of a good thing? In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[22] MA. Jovanovic. Modelling large-scale peer-to-peer networks and a case study of gnutella. Master's thesis, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, June 2000.

[11] MA. Jovanovich, FS. Annexstein, and KA. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical report, ECECS Department, University of Cincinnati, Cincinnati, OH 45221, 2001. Technical Report.

[23] HC. Kim. P2p overview. Technical report, Korea Advanced Institute of Technology, August 2001.

[37] J.Kubiatowicz, D.Bindel, Y.Chen, P.Eaton, D.Geels, SR. Gummadi, H.Weatherspoon, W.Weimer, C.Wells, and B.Zhao. Oceanstore: An architecture for global-scale persistent storage. In Proceedings of ACM ASPLOS. ACM, November 2000.

[10] D.Liben-Nowell, H.Balakrishnan, and D.Karger. Observations on the dynamic evolution of peer-to-peer networks. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[16] C.Lv, P.Cao, ECohen, K.Li, and S.Shenker. Search and replication in unstructured peer-to-peer networks. Preprint, Optional 2001.

[8] Q.Lv, S.Ratnasamy, and S.Shenker. Can heterogeneity make gnutella scalable? In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[24] S.McCarthy, B.Hore, I.Issenin, S.Tauro, and H.Songmei. Survey on p2p file sharing systems.

[25] Computer networking. Karnegie-Melon, Lecture Notes.

[28] The peer-to-peer Workgroup website: http://www.p2pwg.org.

[39] CG. Plaxton, R.Rajaraman, and AH. Richa. Accessing nearby copies of replicated objects in a distributed environment. In Proceedings of ACM SPAA. ACM, June 1997.

[21] S.Ratnasamy, P.Francis, M.Handley, and R.Karp. A scalable content-addressable network. In Proceedings of SIGCOMM 2001, August 2001.

[38] S.Rhea, C.Wells, etal. Maintenance-free global storage in oceanstore. Submission to IEEE Internet Computing, 2001.

[5] M.Ripeanu and I.Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[35] R.Sadasiv. Next generation p2p content networks - syndicating dark matter. In Proceedings of the O'Reilly Peer-to-Peer and Web Services Conference, November 2001.

[13] S.Saroiu, K.Gummadi, and S.Gribble. A measurement study of peer-to-peer file sharing systems. In Proceedings of Multimedia Conferencing and Networking, San Jose, January 2002.

[6] S.Saroiu, PK. Gummadi, and SD. Gribble. Exploring the design space of distributed peer-to-peer systems: Comparing the web, triad and chord/cfs. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[4] A.Serjantov. Anonymizing censorship resistant systems. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), MIT Faculty Club, Cambridge, MA, USA, March 2002.

[33] C.Shirky. What is p2p... and what isn't. O'Reilly Network, November 2000.

[34] C.Shirky, K.Truelove, R.Dornfest, and L.Gonze. 2001 P2P Networking Overview, chapter Chapter 1: All the Pieces of Pie. O'Reilly, October 2001.

[17] I.Stoica, R.Morris, D.Karger, MF. Kaashoek, and H.Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of SIGCOMM 2001, August 2001.

[26] http://groups.yahoo.com/group/the_gdf/files/supernodes.html.

[14] IH. Witten, A.Moffat, and TC. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kauffman, second edition, 1999.

[19] BY. Zhao, J.Kubiatowicz, and AD. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 94720, April 2001.

[30] X.Zhichen, M.Mahalingam, and M.Karlsson. Turning heterogeneity to an advantage in overlay routing. Technical Report HPL-2002-126, HP Labs, 2002.