A Survey of Peer-to-Peer Networks

B. Pourebrahimi K. Bertels S. Vassiliadis
Computer Engineering Laboratory, ITS, TU Delft, The Netherlands
{behnaz, koen, stamatis}@ce.et.tudelft.nl

Abstract—**The limitations of client/server systems become evident in large scale distributed environments. In such systems individual resources are concentrated on one or a small number of nodes and in order to provide access with acceptable response times sophisticated load balancing and fault-tolerance algorithms have to be applied. Also limitation on the network bandwidth adds the bottleneck problem. These problems have motivated researchers to come up with approaches to distribute processing loads and network bandwidth among all nodes participating in a distributed system. Peer-to-Peer systems offer an alternative to traditional client/server systems that solve bottleneck problems but need complex algorithms.**

This paper provides an overview on different p2p architectures and compares them with each other regarding some issues such as scalability, fault tolerance and manageability. P2P systems constitute highly dynamic networks of peers with complex topologies that create an overlay network. P2P applications need sophisticated discovery mechanisms to enable peers to find, identify and communicate with other peers. We discuss the discovery mechanisms in p2p systems which are based on the topology of the network and study the potential challenges in p2p networks such as reliability, security and adaptability.

Keywords: Peer-to-Peer, Distributed systems, Centralized, Decentralized, Resource discovery

I. Introduction

Computation in networks of processing nodes, each holding a part of the inputs and/or resources initially, can be classified into centralized or distributed computations(see figure 1). A centralized solution relies on one node being designated as the computer node that processes the entire application locally. In distributed computation, the processing steps of the application are divided among the participating nodes. The goal in such systems is to minimize communication and computation cost. Distributed systems can be further classified into a client-server model and a P2P model. In the client-server model, the server is the central registering unit, as well as the only provider of content and services. A client only requests content or the execution of services, without sharing any of its own services. The client-server model can be flat where all clients only communicate with a single server or it can be hierarchical for improved scalability.

Peer-to-Peer systems offer an alternative to traditional client-server systems for some application domains. A P2P network is a distributed network composed of a large number of distributed, heterogeneous, autonomous, and highly dynamic peers in which participants share a part of their own resources such as processing power, storage capacity, softwares, and files contents. The participants in the P2P network can act as a server and a client at the same time. They are accessible by other nodes directly, without passing intermediary entities. The P2P models can be pure or hybrid. In pure P2P any single, arbitrary chosen terminal entity can be removed from the network without having the network suffering any loss of network service. Hybrid P2P allows the existence of central entities in its network to provide parts of the offered network services.

There are several concepts underlying p2p systems: sharing resources, decentralization and self organization. Resource sharing implies that applications can not be set up by a single node. Shared resources can be physical resources such as disk space, CPU or network bandwidth, as well as, logical resources such as services or different forms of knowledge. Decentralization is an immediate consequence of sharing of resources. Decentralization is in particular interesting in order to avoid single point of failures and bottlenecks. When a p2p system becomes fully decentralized then there exists no longer a node that can centrally coordinate its activities or a database to store global information about the system centrally. Therefore nodes have to self-organize themselves,based on whatever local information is available and interacting with locally reachable nodes (neighbors). The global behavior then emerges as the result of all the local behaviors that occur.

II. P2P Architectures

Decentralization is one of the major concept of p2p systems. This includes distributed storage, processing, information sharing and also control information. Based on the degree of decentralization in a p2p system, we can classify them into two categories:

A. Purely Decentralized

A pure p2p system is a distributed system without any centralized control. In such systems all nodes are equivalent in functionality. In such networks the nodes are named as ”servant” (SERver+cliENT), the term servent represents the capability of the nodes of a peer-to-peer network of acting at the same time as server as well as a client.

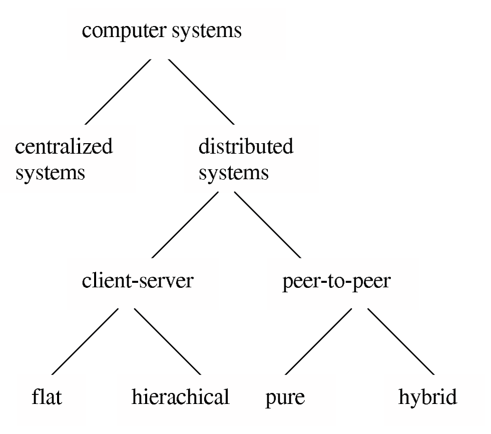

Fig. 1. computer systems

Gnutella[32], Freenet[12], Chord[35] and CAN[30] are instances of such systems.

Pure p2p systems are inherently scalable. Scalability in the system is usually restricted by the amount of centralized operation necessary and such system largely avoid central instances or servers. This kind of systems are inherently fault-tolerant, since there is no central point of failure and the loss of a peer or even a number of peers can easily be compensated. They also have a greater degree of autonomous control over their data and resources. On the other hand such systems present slow information discovery and there is no guarantee about quality of services. Also because of the lack of a global view at the system level, it is difficult to predict the system behavior.

B. Hybrid Architecture

In hybrid P2P systems [41], there is a central server that maintains directories of information about registered users to the network, in the form of meta-data. The end-to-end interaction (data exchange) is between two peer clients. There are two kinds of hybrid systems: centralized indexing and decentralized indexing. In centralized indexing [figure 2] a central server maintains an index of the data or files that are currently being shared by active peers, Each peer maintains a connection to the central server, through which the queries are sent. This architecture is used by Napster[7]. Such systems with the central server are simple and they operate quickly and efficiently for discovery information. Searches are comprehensive and they can provide guarantee in searches. On the other hand they are vulnerable to censorship and malicious attack. Because of central servers they have a single point of failure. They are not inherently scalable, because of limitations on the size of the database and its capacity to respond to queries. As central directories are not always updated, they have to be refreshed periodically.

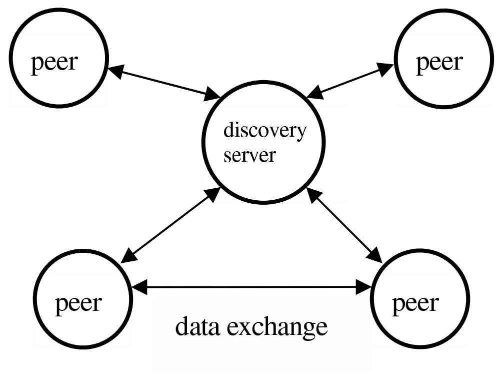

Fig. 2. centralized indexing

In decentralized indexing [figure 3], a central server registers the users to the system and facilitates the peer discovery process. In these systems some of the nodes assume a more important role than the rest of nodes. They are called ”supernodes”[41]. These nodes maintain the central indexes for the information shared by local peers connected to them and proxy search requests on behalf of these peers. Queries are therefore sent to SuperNodes, not to other peers. Kazaa[5][15] and Morpheus[6] are two similar decentralized indexing systems. In such systems peers are automatically elected to become SuperNodes if they have sufficient bandwidth and processing power and a central server provides new peers with a list of one or more SuperNodes with which they can connect.

More recent architectures, such as Gnutella [2] also uses the concept of Super Nodes. 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 SuperPeer is available, it tries to become a SuperPeer again for another probation period.

In comparison with purely decentralized systems, they reduce the discovery time and also they reduce the traffic on messages exchanging between nodes. In comparison with centralized indexing, they reduce the workload on central server but they present slower information discovery. Also in this kind of systems, there is still no unique point of failure as on single central sever. If one or more supernodes go down, the nodes connected to them can open new connection with others, and the network will continue to operate. In the case a large number or even all supernodes go down, the existing peers become supernodes themselves.

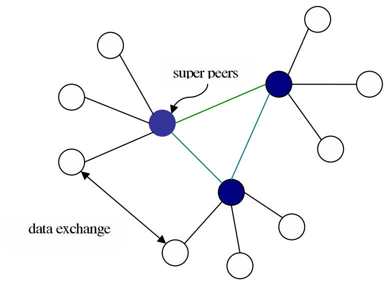

Fig. 3. distributed indexing

III. Discovery mechanisms for p2p systems

Distributed peer-to-peer systems often require a discovery mechanism to locate specific data within the system. P2P systems have evolved from first generation centralized structures to second generation flooding-based and then third generation systems based on distributed hash tables[19]:

- **Centralized indexes and repositories**

This mechanism is used in hybrid systems. In this model the peers of the community connect to a centralized directory servers, which store all information regarding location and usage of resources. Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. The best peer could be the one that is cheapest, fastest, nearest, or most available, depending on the user needs. Then the data exchange will occur directly between the two peers. Napster uses this method [10]. A central directory server maintains: an index with meta data (file name, time of creation etc.) of all 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. In the beginning the client contacts the central server and reports a list with the files it maintains. When 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 4).

- **Flooding broadcast of queries**

This model is a pure p2p model in which each peer does not maintain any central directory and each peer publishes information about the shared contents in the P2P network. Since no single peer knows about all resources, peers in need for resources flood an overlay network queries to discover a resource, each request from a peer is flooded (broadcasted) to directly connected peers, which themselves flood their peers etc., until the request is answered or a maximum number of flooding steps occur. Flooding based search networks are built in an ad hoc manner, without restricting a priori which nodes can connect or what types of information they can exchange[13]. Different broadcast policies have been implemented to improve search in P2P networks[40], [36], [38]. Original architecture of Gnutella[1] uses the flooding broadcast to find the files in the network. It works as a distributed file storage system. There is four types of messages in the Gnutella protocol: 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 to their neighbors. Gnutella originally uses TTL-limited flooding (or broadcast) to distribute Ping and Query messages. At each hop the value of the field time-to-live(TTL) is decremented, and when it reaches zero the message is dropped. In order to avoid loops, the nodes use the unique message identifiers to detect and drop duplicate messages. This approach improves efficiency and preserve network band width. Once a node receives a QuerryHit 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 (see figure 5).
Although the flooding protocol might give optimal results in a network with a small to average number of peers, it does not scale well. Furthermore, accurate discovery of peers is not guaranteed in flooding mechanisms. Also TTL effectively segments the Gnutella network into subsets, imposing on each user a virtual horizon beyond which their messages cannot reach. If on the other hand the TTL is removed, the network would be swamped with requests.

- **Routing Model**

The routing model adds structure to the way information about resources are stored using distributed hash tables. This protocol provide a mapping between the resource identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired resource. This protocol reduces the number of p2p hops that must be taken to locate a resource. The look-up service is implemented by organizing the peers in a structured overlay network, and routing a message through the overlay to the responsible peer [14]. Several proposals have been recently put forth for implementing distributed P2P look-up services :

– Freenet

Freenet [12] provides file-storage service rather than file-sharing service. In this system each peer from the network is assigned a random ID and each peer also knows a given number of peers. When a document is shared on such a system, an ID is assigned to the document based on a hash of the document's contents and its name. Each peer will then route the document towards the peer with the ID that is most similar to the document ID. This process is repeated until the nearest peer ID is the current peer's ID. Each routing operation also ensures that a local copy of the document is kept. When a peer requests the document from the p2p system, the request will go to the peer with the ID most similar to the document ID. This process is repeated until a copy of the document is found. Then the document is transferred back to the request originator, while each peer participating the routing will keep a local copy.

– Chord

Chord [35] is a decentralized p2p lookup protocol that stores key/value pairs for distributed data items. Given a key, it maps key a node responsible for storing the key's value. In the steady state, in an N-node network, each node maintains routing information about $O(logN)$ other nodes, and resolves all lookups via $O(logN)$ messages to other nodes. Updates to the routing information for nodes leaving and joining require only $O(log^2N)$ messages.

– Content Addressable Networks

CAN[30] is a mesh of N nodes in virtual d-dimensional dynamically partitioned coordinate space. Each peer keeps track of its neighbors in each dimension. When a new peer joins the network, it randomly chooses a point in the identifier space and contacts the peer currently responsible for that point. The contacted peer splits the entire space for which it is responsible into two pieces and transfers responsibility of half to the new peer. the new peer also contacts all of the neighbors to update their routing entities. The CAN discovery mechanism consists of two core operations namely, a local hash-based look-up of a pointer to a resource, and routing the look-up request to the pointer. The CAN algorithm guarantees deterministic discovery of an existing resource in $O(N^{1/d})$ steps.

– Pastry

An approach similar to Cord was also used in Pasty [33]. In the Pastry each node network has a unique identifier (nodId) from a 128-bit circular index space. The pastry node routes a message to the node with a nodeId that is numerically closest to the key contained in the message, from its routing table of $O(logN)$, where N is the number of active Pastry nodes. The expected of routing steps is $O(logN)$. Pastry takes into account network locality; it seeks to minimizes the distance messages travel, according to a scalar proximity metric like the number of IP routing hops.

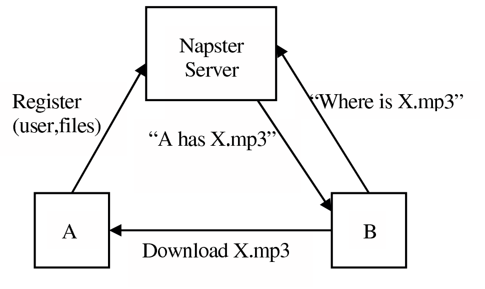

Fig. 4. Resource discovery in Napster

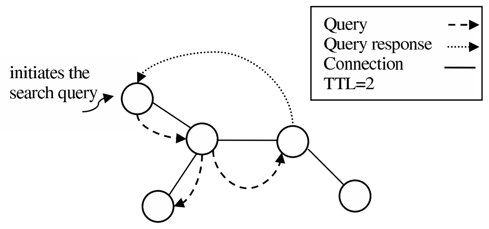

Fig. 5. fboding-based broadcast

IV. P2P NETWORKS STRUCTURE

P2P networks can be classified by the degree to which these overlay networks contain some structure or are created ad-hoc. Structure refer to the way in which the content of the network is located with respect to the network topology. In structured networks, the topology is tightly controlled and the data are placed at specific locations. These systems provide a mapping between the data identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired data. In unstructured networks, the placement of the data is completely unrelated to the overlay topology and peers are connected directly to each other. They are refereed to as neighbors and have no information of each others data. In these systems, searching amounts to random search, in which various nodes are probed and asked if they have any match for the query. For instance, Gnutella is unstructured and Freenet, Chord and CAN are structured.

In many ways, the quality of a P2P system depends on the structural and behavioral properties of its network. Unstructured systems are easy to implement and also they require little maintenance but they lack scalability. As the number of participant peers increases, the number of messages exchanged for a resource search grows. Flooding

	unstructured pure p2p	**structured pure p2p**	**centralized indexing hybrid p2p**	**distributed indexing hybrid p2p**
Scalable	no	yes	no	yes
Flexible	yes	no	no	yes
Robustness	yes	yes	no	yes
Manageable	no	yes	yes	yes

TABLE I
COMPARISON OF P2P MECHANISMS

search protocol used in unstructured P2P networks is very sensitive to the number of edges in the network graph. If the number of links is to small, all nodes will not be reachable in a reasonable amount of time. Conversely, if there are too many links, numerous identical copies of the query message will arrive at many nodes from different directions, resulting in wasted bandwidth. In structured P2P systems peers maintain information about what resources neighboring peers offer. It increases the cost of maintenance efforts during changes in the overlay network when peers join or leave.

V. P2P APPLICATIONS

The domains of p2p applications can be classified into four categories[25]:

- **File sharing**

Content storage and exchange is one of the areas where P2P technology has been most successful. File sharing applications [10], [20], [34] focus on storing information on and retrieving information from various peers in the network. One of the best-known example of such p2p systems is Napster, it became famous as a music exchange system. Other instances are Gnutella, Freenet, Kazaa, Chord, etc.

- **Distributed computing**

These applications use resources from a number of networked computers. The general idea behind these applications is that idle cycles from any computer connected to the network can be used for solving the problems of the other computers that require extra computation. SETI@home is one example of such systems. SETI (Search for Extraterrestrial Intelligence) [9] is a scientific search project aimed at building a huge virtual computer based on the aggregation of the computer power offered from internet-connected computers during their idle periods. The project uses two major components: the database server and the client. Clients can help with search for extra-terrestrial life by running the search program for a specified portion of the universe. This project strongly relies on its server to distribute jobs to each participating peer and to collect results after processing is done.

- **Collaboration**

Collaborative p2p applications aim to allow application-level collaboration between users. These applications range from instant messaging and chat, to on line games, to shared applications that can be used in business, educational, and home environments. Such as Groove, Jabber. Jabber [4] is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real time.
Groove [3] provides a variety of applications for communication, content sharing (files, images and contact data), and collaboration (i.e. group calendaring, collaborative editing and drawing, and collaborative Web browsing).

- **Platforms**

P2P platforms provide infrastructure to support distributed applications using p2p mechanisms. P2P components used in this context are for instance naming, discovery, communication, security and resource aggregation. JXTA [8] is p2p platform that provides a general-purpose network programming and computing infrastructure. It creates a p2p system by identifying a small set of basic functions necessary to support p2p applications and providing them as building blocks for higher-level functions. it includes three layer: core, services and applications. JXTA core provides core support for peer-to-peer services and applications. At the core, capabilities must exist to create and delete peer groups, to advertise them to potential members, to enable others to find them, and to join or leave them. At the next layer, the core capabilities can be used to create a set of peer services, including indexing, searching, and file sharing. In the third layer peer applications can be built using these facilities[28].

VI. CHALLENGES IN P2P SYSTEMS

Peer-to-peer systems offer a number of advantages over conventional client-server systems such as scalability, fault tolerance, performance. However, there are some challenges that these systems are dealing with:

A. Security

Distributed implementations create additional challenges for security compared to client-server architecture. Since in peer-to-peer systems the set of active peers is dynamic and also peers don't trust each other, achievement a high level of security in peer-to-peer systems is more difficult than non-peer-to-peer systems. Traditional security mechanisms to protect data and systems from intruders and attacks such as firewalls can not protect peer-to-peer systems since they are essentially globally distributed and also these mechanisms can inhibit peer-to-peer communication. Therefore new security concepts are required that allows interaction and distributed processing in peer-to-peer systems. [39]

B. Reliability

A reliable system is a system that can be recovered when a failure occurs. The factors which should be taken into account for reliability are data replication, node failure detection and recovery, existence of multiple guarantees for location information to avoid a single point of failure and the availability of multiple paths to data. Data replication increases reliability by increasing redundancy and locality. There are two strategies for replication, owner replication and path replication. In owner replication, when the search is successful, the data is stored at the requester node only. In path replication, when a search succeeds, the data is stored in all nodes along the path from requester node to provider node [22]. P2P communities can also replicate and replace the data as a function of their popularity to achieve satisfactory performance[17].

In structured peer-to-peer overlay networks the messages are routed in a small number of hops using small per-node routing state. The overlays should update routing state automatically when nodes join or leave, and it should route messages correctly even when a large fraction of the nodes crash or the network partitions. To achieve reliability in such systems, nodes must consume network bandwidth to maintain routing state, so to reduce this cost the techniques should be employed that adapt to operating condition[23]. For increasing fault-tolerance and reliability in unstructured P2P systems, dynamically adding redundant links to the system has been addressed [24]. In this way no single disconnection should cause disconnection of system or too much increase of routing steps.

C. Flexibility

One of the important aspects in P2P systems is the autonomy of peers so that they can join and leave at their will. Recent peer-to-peer (P2P) systems are characterized by decentralized control, large scale and extreme dynamism of their operating environment. To deal with the scale and dynamism the properties of adaptation and self-organization are required to be considered in building p2p systems.

More recent unstructured P2P systems, like KaZaA and GIA [11] address the dynamic nature of the environment. In Kazaa queries are forwarded only to supernodes, which maintain a list with the file names of their connected peers, avoiding overloading all peers of the system. GIA is a Gnutella like system which aims to respond to high aggregate query rates. In GIA each peer calculates the maximum number of queries it can handle per second and based on this metric the number of neighbors to which the peer can connect or forward a request is computed [16].

In standard structured P2P systems, static identifiers are assigned to peers and distributed data structures are constructed based on these identifiers, so the overlay network structure is determined by the choice of these identifiers and in turn any self-organization of the system is prevented. Structured systems based on distributed hash tables (DHTs) should perform lookups quickly and consistently while nodes arrive and leave the system [31], [21]. For instance Chord [35] adapts as nodes join and leave the system, and answer queries even if the system is continuously changing. Discovering that a node has joined is achieved through a self-stabilization protocol that every node runs periodically[37].

Complex Adaptive Systems (CAS) which commonly used to explain the behavior of certain biological and social systems can be used as a model to build adaptive P2P networks[27].

D. Load Balancing

Distribution of the data to be stored or computations to be carried out by the nodes is a critical issue for the efficient operation of peer-to-peer networks. A particular method for such distribution in peer-to-peer systems is the distributed hash table (DHT), in which each data item that is stored is mapped to unique identifier ID. The identifier space is partitioned among the nodes and each node is responsible for storing all the items that are mapped to an identifier in its portion of the space. In such approaches load balancing should be considered in both address-space balancing that balances the distribution of the key address space to nodes and item balancing in the case that distribution of items in the address space can not be randomized. In this method, each node is free to migrate anywhere and it has no restriction to be in a certain number of virtual node locations (it means the items can migrate among the nodes) [18], [42], [29].

Load balancing among the computing nodes in p2p systems can also be implemented by agent-based self-organization models. Messor [26] is a Anthill load balancing algorithm. In Messor, ants adapt their behavior to the load conditions, wandering about randomly when the load is uniformly balanced and moving rapidly towards regions of the network with highly unbalanced loads. They are resilient to failures as jobs assigned to crashed nodes are simply reinserted in the network by the nest that generated them and they are self-organized as new nests or nodes may join to the system, and their computing power is rapidly exploited to carry on the computation, as soon as ants discover the nest and start to assign it jobs transferred from other nests.

VII. Conclusion

Considering different architectures of peer-to-peer systems, system designers should evaluate the requirements for their particular applications and choose a topology for the platform that matches their needs. To compare them briefly (see table I), we can say that in pure peer-to-peer networks every peer is given equal responsibility irrespective of its computing/network capabilities, this can lead to reduction of performance as less capable nodes are added. Pure p2p systems lack manageability since every peer is its own controller. Unstructured pure p2p systems in which blind flooding search is used are not salable since in large scale systems the large number of exchange messages limits the scalability. Using structured systems or intelligent search approaches can solve scalability limitation. The disadvantage of standard structured systems is that it is hard to maintain the structure required for routing in a very transient node population, in which nodes join and leave at a high rate. It should be taken into account that some structured systems like Chord have overcome to this problem and they can adapt efficiently as nodes join and leave the system.

Pure p2p systems are fault tolerant, since failure of any particular node does not impact the rest of the system. Hybrid p2p systems solve the manageability problem of pure p2p systems, so that the control server/servers acts as a monitoring agent for all the other peers and ensures information coherence. regarding distributed indexing and centralized indexing systems, drawbacks associated to centralized indexing systems are single point of failure when central server goes down and also not being scalable because of capacity of server to maintain database and to respond to queries. Distributed indexing systems alleviate these shortcomings by using super-peers. Although super-peer clusters are efficient, scalable and manageable, in order to avoid a single point of failure for the clients in a cluster, some policies of super-peer redundancy should be taken into account. As in the case of fail over super-peer, these strategies should be able to take over the job of the primary super-peer.

References

[1] The gnutella protocol specifi cation v0.4.

[2] Gnutella2, http://www.gnutella2.com.

[3] Groove, http://www.groove.net.

[4] Jabber, http://www.jabber.org.

[5] Kazaa, http://www.kazaa.com.

[6] Morpheus, http://www.morpheus.com.

[7] Napster, http://www.napster.com.

[8] Project jxta, http://www.jxta.org.

[9] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[10] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer fi le sharing technologies, 2002.

[11] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418. ACM Press, 2003.

[12] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[13] B. F. Cooper and H. Garcia-Molina. Ad hoc, self-supervising peer-to-peer search networks. Technical report, 2003.

[14] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller. Hierarchical p2p systems. In *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, Klagenfurt, Austria, 2003.

[15] Nathaniel S. Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p fi le-sharing. In *CHI '03: Proceedings of the conference on Human factors in computing systems*, pages 137–144. ACM Press, 2003.

[16] Evangelia Kalyvianaki and Ian A. Pratt. Building adaptive peer-to-peer systems. In *Peer-to-Peer Computing*, pages 268–269, 2004.

[17] Jussi Kangasharju, Keith W. Ross, and et al. Adaptive content management in structured p2p communities.

[18] David R. Karger and Matthias Ruhl. Simple effi cient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43. ACM Press, 2004.

[19] Mandar Kelaskar, Vincent Matossian, Preeti Mehra, Dennis Paul, and Manish Parashar. A study of discovery mechanisms for peer-to-peer applications. In *CCGRID*, pages 444–445, 2002.

[20] Ulrike Lechner. Peer-to-peer beyond fi le sharing. In *IICS*, pages 229–249, 2002.

[21] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242. ACM Press, 2002.

[22] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM Press, 2002.

[23] Ratul Mahajan, Miguel Castro, and Antony Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *IPTPS'03*, February 2003.

[24] Leonardo Mariani. Fault-tolerant routing for p2p systems with unstructured topolog. In *proceedings of the 2005 International Symposium on Applications and the Internet (SAINT 2005), IEEE Computer Society*, February 2005.

[25] Characteristics Andreas Mauthe. Peer-to-peer computing: Systems, concepts and.

[26] A. Montresor, H. Meling, and A. Montresor. Messor: Load-balancing through a swarm of autonomous agents, 2002.

[27] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Towards adaptive, resilient and self-organizing peer-to-peer systems. *Lecture Notes in Computer Science*, 2376:300–??, 2002.

[28] Ra Ti On. Project jxta: An open, innovative collaboration.

[29] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard M. Karp, and Ion Stoica. Load balancing in structured p2p systems. In *IPTPS*, pages 68–79, 2003.

[30] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network, 2000.

[31] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.

[32] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[33] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[34] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer fi le sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[35] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[36] In suk Kim, Yong hyeog Kang, and Young Ik Eom. An effi cient contents discovery mechanism in pure p2p environments. In *GCC (1)*, pages 420–427, 2003.

[37] C. Tryfonopoulos and M. Koubarakis. Distributed resource sharing using self-organized peer-to-peer networks and languages from information retrieval. In *Proceedings of International Workshop on Self-* Properties in Complex Information Systems (SELF-STAR)*, Bertinoro, Italy, 2004.

[38] Dimitrios Tsoumakos and Nick Roussopoulos. A comparison of peer-to-peer search methods. In *WebDB*, pages 61–66, 2003.

[39] Dan S. Wallach. A survey of peer-to-peer security issues. In *ISSS*, pages 42–57, 2002.

[40] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 5. IEEE Computer Society, 2002.

[41] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceeding of 19th International Conference on Data Engineering*, page 49, 2003.

[42] Yingwu Zhu and Yiming Hu. Effi cient, proximity-aware load balancing for structured p2p systems. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 220. IEEE Computer Society, 2003.