Evolution of Peer-to-peer algorithms: Past, present and future.

Alexei Semenov
Helsinki University of Technology
alexei.semenov@hut.fi

Abstract

Today peer-to-peer applications are widely used for different purposes. People chat, talk, play games, share resources and content using peer-to-peer technology every day. Such peer-to-peer applications as Skype (P2P telephony), ICQ (P2P instant messaging), Kazaa (P2P content sharing) and SETI@home (P2P resource sharing) have enjoyed huge popularity in recent years and are used by millions of people around the world.

Peer-to-peer applications played an important role in the evolution of the Internet. The evolution path of P2P algorithms and applications is interesting and studying it can tell us the direction in which the evolution will proceed in the future. In the current paper we intend to study this evolution by reviewing the existing literature and analysing it. We will overview historical developments in this field, as well as ongoing research on the topic, then analyse our findings and try to figure out the future trends in this field.

KEYWORDS: P2P, peer-to-peer, evolution

1 Introduction

In recent years Peer-to-peer (P2P) has been the main reason for the growth of Internet traffic. According to Mellin [6], at the moment P2P traffic contributes to more than 50% of the total Internet traffic and to about 80-90% of the local traffic, and is growing with the rate of 250% per year. This makes P2P an interesting and important topic to understand.

Back in the late 1960s Internet was created as a peer-to-peer system with the goal of sharing computing resources across the USA. It was used by a relatively small group of people and thus was very open and free. Everyone in this Internet community was working on building a reliable, efficient and powerful network. FTP and Telnet, the most popular applications back then were based on client/server architecture [5]. 1990s were the years of serious changes for the Internet, millions of new people started using it, bringing new applications and changing the way the network was used. Also a lot of malicious users have joined the network and spam became a huge problem. Because of these changes in the user community, a lot of firewalls were deployed and asymmetric links such as ADSL or cable modems were growing. In the year 2000 a new kind of peer-to-peer applications have emerged. P2P file sharing systems quickly became very popular and played an important role in the latest evolution of the Internet.

There are many different definitions of P2P. Schollmeier [8] defines Peer-to-peer network as "A distributed network architecture, where the participants of the network share a part of their resources, which are accessible by other peers directly, without passing intermediary entities." This definition is backed up by Milojicic [7], saying that resources can be of various types, meaning that P2P systems can be divided into four categories: distributed computing, file sharing, collaborative systems and P2P platforms.

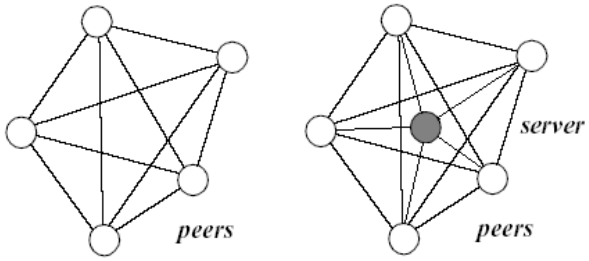

Figure 1: Examples of pure and hybrid P2P systems [7]

Although P2P has been invented a long time ago, it has not received attention of the wide audience until the appearance of P2P file sharing systems. That is why the focus of this paper will be only on file sharing P2P systems. We will not look at legal issues concerning P2P or impact of P2P on the carriers, but rather concentrate on P2P algorithms and their functions, especially forming an overlay network (joining and leaving of nodes), and searching for content.

Schollmeier [8] divides file sharing systems into two groups: pure P2P systems and hybrid P2P systems. In pure P2P systems all peers are equal and each peer is acting both as a client and as a server. In such systems any peer can be removed from the network without affecting operation of the network. Gnutella and FreeNet are examples of pure P2P systems. On the other hand, in hybrid P2P systems majority of peers are equal, but some peers have special functions and are called servers. E.g. in Napster, the first and widely known P2P file sharing system, these special peers were used for searching purposes. Fig. 1 shows examples of pure and hybrid peer-to-peer systems. In Kazaa an intermediate solution was introduced in the form of super-peers, which contain information that normal peers may not have [10].

The structure of this paper is as follows. In Sec. 2, we look in detail at different Peer-to-peer applications and algorithms from the first ones to appear to the latest ones, including such applications as Napster, Gnutella, FreeNet, Kazaa and algorithms behind them. In Sec. 3, we compare these applications and algorithms in order to find their differences and commonalities. In Sec. 4, we look at the current ongoing research and evaluate its importance for the next generations

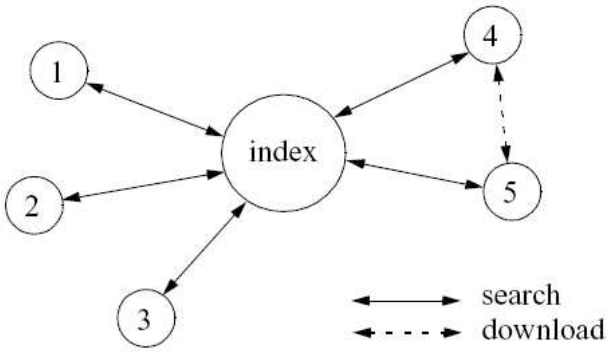

Figure 2: Centralized directory model [7]

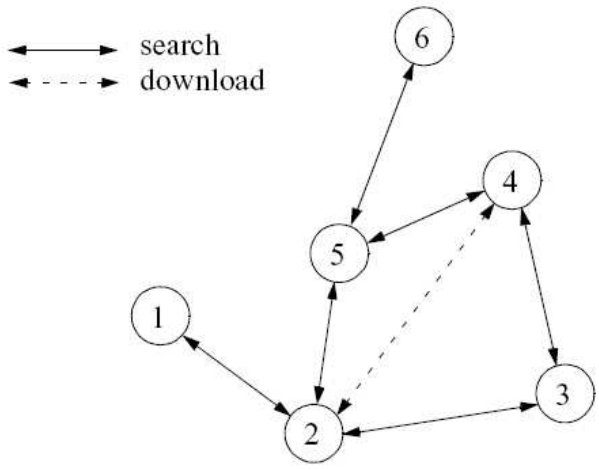

Figure 3: Flooded requests model [7]

of peer-to-peer applications.

2 Evolution of peer-to-peer: algorithms, features and applications

In this section, we look at different generations of algorithms used in file sharing Peer-to-peer applications. Sec. 2.1 discusses applications and algorithms based on centralized directory model and will be refered to as 1st Generation. Sec. 2.2 discusses applications and algorithms based on flooded requests model and will be refered to as 2nd Generation. Finally, sections 2.4-2.7 will discuss applications and algorithms based on document routing model and refered to as 3rd Generation.

2.1 1st Generation

Centralized directory model was the first peer-to-peer model that was widely used. It is used in hybrid P2P systems, because it perfectly fits the definition: majority of peers are equal, but some peers have more advanced functions and are called servers [7]. Fig. 2 shows an example of centralized directory model. This is how this algorithm works: peers inform the server about the files they are willing to share and server stores location information of the shared files. When a peer wants to get some file it sends a request to the server. Using location information that it has gathered, the server finds peers containing the requested file and matches the request with the best suited peer based on connection speed, size of the file or some other parameter. After that peers do not need the server anymore and can negotiate file exchange directly between themselves. Centralized directory model was used in Napster, probably the most well known P2P file-sharing application.

The main disadvantage of centralized directory model comes from the fact that when the number of peers grows, the size of the servers must also grow and this implies a scalability limit. But as we can see from Napster's experience, this model works really well in most of the aspects and its main weakness concerns copyright issues.

2.2 2nd Generation

Flooded requests model appeared next. In this model there do not exist any servers and all peers are equal, which makes systems using this model to be pure P2P systems [7]. Here, peers do not inform anyone about the content they are willing to share, instead when a peer wants to locate some file, it sends a request to those peers it is directly connected to, if the requested file is not found among them, each peer resend the request to peers it is connected to, etc. This resending of request is called flooding and the common number of flooding steps varies from 5 to 9. Fig. 3 shows an example of flooded requests model.

Flooded requests model is used in Gnutella. In this application in order to join the system, the peer has to connect to one of several known hosts which forward information about IP and port address to other Gnutella peers [11]. After that, peer can start sending requests. Gnutella has shown to be popular, especially when the number of peers is relatively small. However when the number of peers grows, flooded requests model starts to consume a lot of bandwidth reducing scalability of the system. This is one of the problems, discovered in Gnutella. Another problem concerned "Freeriding". Freeriding means downloading files from other peers but not sharing them. According to research by Adar [11], more than 70% of Gnutella users are free riders. Such behavior results in a smaller number of files available in the system, and decreases the level of anonymity, since the number of peers possessing the files decreases considerably and these peers might be considered to act like "servers". Freeriding is a problem not particulary in Gnutella, but in all P2P applications, that do not measure upload/download ratio.

In order to deal with the disadvantages of Gnutella, a more sophisticated version of flooded requests model was developed. This improved version included so called "super-peers". Introduction of super-peers has led to lower network bandwidth and improved scalability [7]. The most well known application that uses flooded request model with super-peers is Kazaa, which is one of the most used P2P applications today [21]. In Kazaa users that have powerful computers and fast Internet connections are assigned to

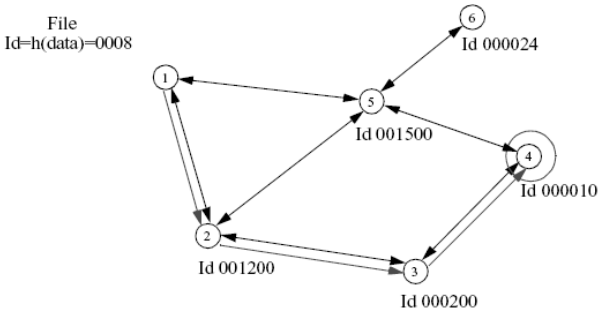

Figure 4: Document routing model [7]

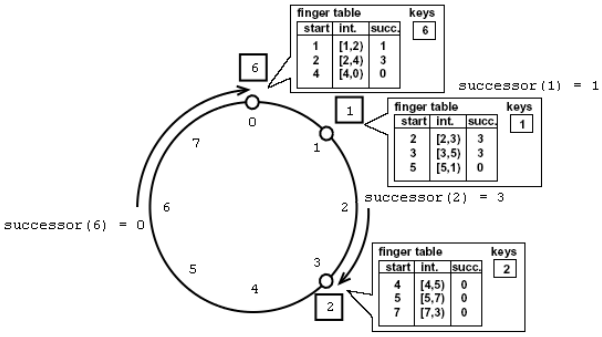

Figure 5: Chord: Identifier circle and finger tables [1]

be supernodes. Other peers inform supernodes about files they are sharing and when someone makes a request, it is not flooded from peer to peer, but it is flooded from supernode to supernode, increasing scalability of the system. In order to reduce freeriding, Kazaa introduces the idea of participation levels. The idea is very simple: the more files other peers download from you, the higher is your participation level. When several peers request the same file, peer with highest participation level will get the priority.

2.3 3rd Generation

Document routing model is the most recent model. It is used in pure P2P systems, where all peers are equal [7]. In this model each peer is assigned an ID and each peer knows several other peers. When a peer wants to share a file, the name and the contents of the file are hashed, producing an ID which is assigned to that file. Then the file is routed to the peer, whose ID is most similar to the file ID. This way, each peer is responsible for a range of file IDs. When a peer wants to get some file, it sends a request containing the ID of the file. This request is forwarded to the peer whose ID is most similar to the file ID. After the file has been located it is transferred to the peer that has requested it. Each peer that participates in either of the routing operations keeps a local copy of the file. Fig. 4 shows an example of document routing model. Document routing model is used in Freenet, software application that has been around for several years, but is still under development.

Document routing model has proved to be effective in networks with large number of peers. But the problem with this model comes from the fact that ID of the requested file must be known before sending a request, which makes the search more difficult to implement. A peer that wants to locate a specific file has to know what information was used by the peer that is sharing the file when the ID of the file was created. Hashing different filename or contents will result in a different ID and the required file will not be found. Another problem is so called "islanding problem", when peers split into groups with no connection between each other [7]. In the following 4 sections we will look at 4 different algorithms that implement the document request model: Chord, Tapestry, Pastry and CAN. Although they differ between each other, they all have the same goal: decreasing the number of hops needed to locate the requested file.

2.4 3rd Generation: Chord

Chord supports only one operation: it takes a key and maps the key onto a node that is responsible for this key. Chord can be seen as a basis of data location algorithm, which can be built on top of Chord. In order to do this a key can be associated with a data item and the key/data item can be stored at the node [1]. In Chord load balance is achieved by spreading the keys evenly over nodes and making all nodes equal, which also leads to decentralization. Chord is also very scalable, because the cost of lookup is *logN*, where *N* is the number of nodes. Chord has been implemented in such real-life applications as CFS [15] and Mnemosyne [16].

Chord uses a combination of consistent hashing [20] and scalable key location. In consistent hashing, each node is assigned an *m*-bit identifier by hashing the node's IP address using a base hash function (e.g. SHA-1). Each key is assigned an *m*-bit identifier by hashing the key itself. Keys are assigned to the nodes using the following rule: identifiers form an identifier circle modulo 2^m. Key *k* is assigned to the first node whose identifier is equal or follows *k* in the identifier circle. The node is called the successor node of *k*, while *k* is called a predecessor of that node. Fig. 5 shows an example of identifier circle with nodes 0, 1 and 3. Here *m* equals 3. Key 1 would be located at node 1, key 2 would be located at node 3, and key 6 at node 0. But using consistent hashing alone might result in a situation, when all nodes would have to be checked in order to find the needed mapping, which results in low scalability. To avoid this scalability problem, Chord uses scalable key location by maintaining additional routing information. The idea behind scalable key location is that each node has a routing table of at most *m* entries, which is called a finger table. An entry in such table includes Chord identifier and an IP address of the relevant node. This way every node does not have to know about all other nodes, but only about small number of neighboring nodes. An example of finger tables can be seen on Fig. 5.

Chord enables nodes to join and leave at any time. When a new node joins the network, it is assigned all keys that were assigned to its successor. When a node leaves the network, all keys that were assigned to it are now assigned to its successor. In order to deal with concurrent joins, Chord uses a stabilization protocol that checks whether the successor pointers are up to date. Stabilization is also used for managing failures of nodes. When a failure occurs, successor

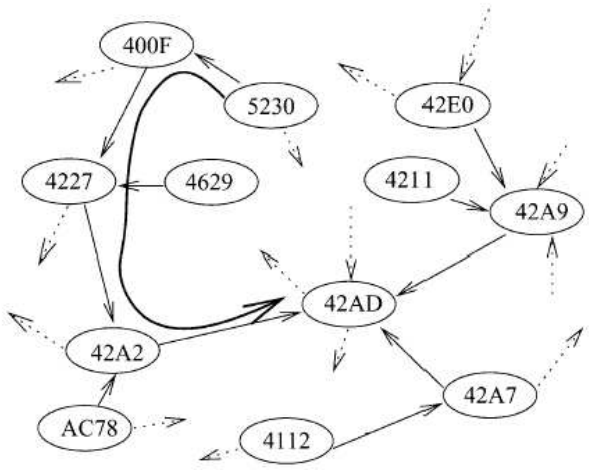

Figure 6: Tapestry: path of a message [2]

pointers have to be updated and this is only possible if they are maintained correctly.

2.5 3rd Generation: Tapestry

Tapestry is an infrastructure that provides decentralized object location and routing (DOLR). Similar to Chord, Tapestry routes messages to nodes based on the keys assigned to the messages [2]. What distinguishes Tapestry from Chord is the introduction of location mapping, which decreases the search time. Location mapping is described in the following paragraphs. Tapestry has been implemented in such real-life applications as Bayeux [19], Mnemosyne [16] and OceanStore [18].

In Tapestry each node and object (e.g. file) are assigned unique ids using a hash function (e.g. SHA-1). Each node has a routing table, which lists IDs and IP addresses of the nodes that it communicates with. These nodes are called its neighbors and the routing table is called neighbor table. DOLR communication is packet-based. It supports 4 types of operations: *PublishObject* publishes the object on the local node, *UnpublishObject* removes location mappings, meaning that the object is no longer shared, *RouteToObject* routes message to the location of the object, *RouteToNode* routes message to the node. Routing is performed digit by digit and at each node possible IDs are chosen from a neighbor table of this node. Fig. 6 shows an example of the path taken by a message with origin at node 5230 and destination at node 42AD.

When an object is published by a node, a mapping message is sent from this node to the node whose ID is most similar to the ID of the published object. Each node on the path of the message stores a location mapping, instead of a copy of the object like in Chord. When some node wants to locate an object, it does not have to find the node whose ID is most similar to the ID of the object in question, it is enough to find any node that stored the location mapping. Tapestry proves to be scalable, with the cost of lookup of $log_b N$, where *N* is the number of nodes and *b* is the base of the IDs.

In Tapestry nodes can leave or join at any time. When a new node joins the network, it is assigned responsibility for objects with most similar IDs, a routing table is created for this node and routing tables of other nodes are updated. When a node leaves the network voluntarily, responsibilities are reassigned and routing tables are updated. To manage the situation when the node leaves involuntarily, Tapestry provides redundancy in its routing tables. It also uses periodical republishing of object references.

2.6 3rd Generation: Pastry

Pastry is a distributed object location and routing infrastructure, where messages are routed to nodes based on the provided keys [3]. Pastry has been implemented in such real-life applications as Scribe [14] and PAST [17].

In Pastry each node is assigned a unique128bit ID upon joining the network. IDs are assigned randomly and are roughly evenly distributed across the id space. IDs are generated by hashing the IP address or public key of the node using SHA-1 hash function. Each node has a routing table, a neighborhood set and a leaf set. Routing table contains IP addresses of the nodes, whose IDs have the same first *n* digits, where *n* is the row number in the routing table. Neighborhood set lists IP addresses and IDs of 2^b nodes that are closest to the node in question, where *b*is the base of the ID. Usually *b* equals 4 or 5. Leaf set lists IDs of $2^b - 1$ nodes, whose IDs are numerically smaller than the ID of the node in question and IDs of $2^b - 1$ nodes, whose IDs are numerically larger than the ID of the node in question. Routing table and leaf set are used in routing messages, while neighborhood set is used in maintaining locality. When a node receives a message it needs to route, it first checks its leaf set and only then a routing table. As in Tapestry, routing is performed digit by digit and at the end the message arrives at the node with an ID that is most identical to the key. Pastry proves to be scalable, with the cost of lookup of $log_{2^b} N$, where *N* is the number of nodes. The variable *b* is responsible for the tradeoff between the size of the routing table and the number of hops needed to locate the node.

Nodes can join or leave the Pastry network at any time. When a node joins or leaves, the routing tables and leaf sets of this and other nodes are updated. To control concurrent joins or leavings of nodes, Pastry uses timestamps that are attached to messages and are checked when routing tables and leaf sets are updated. To solve the possible "islanding problem", Pastry uses IP multicast searches for isolated overlays, which also improves the routing tables.

2.7 3rd Generation: Content-Addressable Networks (CAN)

Content-Addressable Network (CAN) is a distributed infrastructure that maps keys onto nodes [4]. CAN was implemented in a real-life application called CAN-MC [13]. The design of CAN is based on a *d*-dimensional Cartesian coordinate space, which is dynamically divided between all nodes of the system at any point of time. The more dimensions there are in coordinate space, the shorter is the length of the routing path. Fig. 7 shows a 2-dimensional coordinates space, that was divided between 5 nodes. The key of the (key, value) pair is mapped onto a point in the space by using

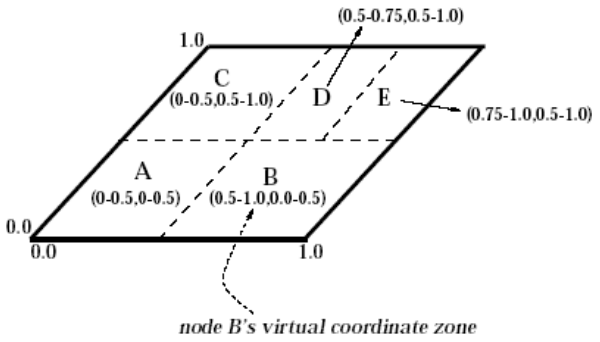

Figure 7: CAN: 2-dimensional space with 5 nodes [4]

P2P System	Algorithm Comparison Criteria		
	Parameters	Hops to locate data	Reliability
Napster	none	constant	Central server returns multiple download locations, client can retry
Gnutella	none	no guarantee	receive multiple replies from peers with available data, requester can retry
Chord	N - number of peers in network	$\log N$	replicate data on multiple consecutive peers, app retries on failure
CAN	N - number of peers in network d - number of dimensions	$d \cdot N^{1/d}$	multiple peers responsible for each data item, app retries on failure
Tapestry	N – number of peers in network b – base of the chosen identifier	$\log_b N$	replicate data across multiple peers, keep track of multiple paths to each peer
Pastry	N - number of peers in network b - base of the chosen identifier	$\log_b N$	replicate data across multiple peers, keep track of multiple paths to each peer

Figure 8: Comparison of different P2P location algorithms [7]

a uniform hash function. Then (key, value) pair is stored at the node who "owns" the part of the coordinate space, where the point lies. This part of the coordinate space is called a zone.

Each node has a routing table, which lists an IP address and a virtual coordinate zone of each of its neighbors in the coordinate space. Each node routes a message to the neighbor whose coordinates are the closest to the coordinates of the destination node.

When a new node joins the network, it finds an existing node, the zone of this node is divided into two parts, one of these parts is assigned to the new node and the other part is kept by the already existing node. After that the routing tables of the neighbors of this node have to be updated. When a node leaves the network, the zone that was assigned to it is reassigned to one of its neighbors: either the one whose zone in combination with the leaving node's zone can produce a single valid zone, or the one with the smallest zone. This node will then temporarily handle two zones until the background zone-reassignment algorithm reassigns the zones between existing nodes in such way that each node will have only one zone. Failures of nodes are handled using a takeover algorithm that assigns the zone of the failed node to one of its neighbors.

CAN also provides an opportunity to maintain several coordinate spaces, that are independent of each other, with each node of the network having zones in each coordinate space. Introducing multiple coordinate spaces improves data availability and routing fault tolerance. A more advanced CAN design allows a single zone to be shared by several nodes, called peers. This is called zone overloading. These peers have not only a routing table, but also a list of its peers: nodes from the same zone and one neighbor node from the peers in each of its neighbor zones. Zone overloading reduces path length, per-hop latency and improves fault tolerance

3 Analysis

Hybrid P2P systems are easier to manage than pure P2P systems. They also have less bandwidth consumption than pure P2P systems. On the other hand hybrid P2P systems do not scale and have single points of failure because of the centralized model, while pure P2P systems provide robustness and availability through replication as well as self-organization and degree of anonymity. Algorithms used in Napster and Gnutella do not guarantee that an existing object can be found from the first try. In Napster a server might be down, while in Gnutella the number of flooding steps might not be enough to locate the file. DHT-based algorithms provide such guarantees. Fig. 8 presents a comparison of different P2P algorithms.

DHT-based algorithms are similar in features of good scalability, decentralization, load balance and self-organization. However they differ in some aspects. Chord provides flexible naming and better handling of concurrent joins and failures of nodes. Chord can be used in large-scale distributed applications such as cooperative file sharing, time-shared available storage systems and large-scale distibuted computing platforms [1]. Chord, Tapestry and Pastry have path lengths of $logN$ hops, while CAN has a longer path length of $dN^(1/d)$ which makes it less suitable for such applications as P2P Telephony, where a user does not want to wait a long time before a call session is established. CAN's longer path length also makes it less appropriate for P2P instant messaging, because the idea of quick communication is even contained in the name of this service. However CAN might be useful in large scale storage management systems that require a scalable indexing mechanism that supports efficient insert and retrieval of content [12].

4 Future trends

All researchers working on the presented algorithms have stated that their algorithms need improvements in one area or another. E.g. in Chord lookup latency and availability of data in case of existence of faulty Chord participants need to be improved [1]. Availability of data in case of existence of mallicious node also needs to be improved in CAN, in addition to extension of CAN to handle mutable content [4].

Ratnasamy [9] presents a different idea, saying that instead of comparing the existing algorithms, new and more advanced algorithms can be introduced by combining and upgrading the existing algorithms. According to Ratnasamy there are several issues that need to be addressed. The first one is a state-efficiency tradeoff. Is it possible to combine

the existing algorithms in such way that short pathlengths are achieved by using routing tables of small sizes? The second issue concerns resilience to failures. Different scenarious of failures exist, and each of them needs to be analized in order to know how long it takes each algorithm to recover from a failure. The third issue concerns routing hotspots - nodes that are overloaded with routing traffic. Another issue concerns geographic proximity of nodes. Finally an issue of heterogeneity has to be addressed. Current algorithms assume that all nodes have the same bandwidth capacity, but in reality the capacities of different nodes might vary considerably.

5 Conclusions

Huge popularity of P2P applications proves that there is a place for them in the future. P2P algorithms are the hearts of these applications. In this paper we studied the evolution of peer-to-peer algorithms and applications. Based on this information we compared different algorithms on the basis of different features (scalability, simplicity, etc.). We found out that although the difference between DHT-based algorithms is not very big, difference between P2P algorithms of various generations is clearly visible. This indicates that the evolution of P2P algorithms is really taking place. We also looked at the applicability of the algorithms to different types of P2P applications based on the features of the algorithms. Finally we looked at future trends in this area that aim to improve the existing algorithms or to build new algorithms on the basis of existing ones.

References

[1] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 01 Conference*, San Diego, California, Aug 2001.

[2] ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.

[3] DRUSCHEL, P., AND ROWSTRON, A. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov 2001.

[4] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 01 Conference*, San Diego, CA, Aug 2001.

[5] MINAR, N., et al. Peer-to-peer: Harnessing the Power of Disruptive Technologies March 2001.

[6] MELLIN, J. Peer-to-Peer Networking - Phenomenon and impacts to carriers. Presentation at Telecom Forum, Helsinki University of Technology, Sep 2004.

[7] MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. Peer-to-Peer Computing. HP Laboratories Palo Alto, HPL-2002-57, 2002.

[8] SCHOLLMEIER, R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P01)*, 2002

[9] RATNASAMY, S., SHENKER, S., AND STOICA, I. Routing algorithms for DHTs: some open questions. In *Proceedings of the IEEE International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, Mar 2002.

[10] YANG, B., AND GARCIA-MOLINA, H. Comparing Hybrid Peer-to-Peer Systems. Stanford University, Computer Science Department, 2000.

[11] ADAR, E., AND HUBERMAN, B., A. Free Riding on Gnutella, Sep 2000.

[12] LUA, E., K., CROWCFOFT, J., PIAS, M., SHARMA, R., AND LIM, S. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE Communications Survey and Tutorial, Mar 2004.

[13] RATNASAMY, S., HANDLEY, M., KARP, R. AND SCHENKER, S. Application-level multicast using content-addressable networks In *Proceedings of NGC*, London, U.K., Nov 2001.

[14] ROWSTRON, A., KERMARREC, A., M., DRUSCHEL, P. AND CASTRO, M. SCRIBE: The design of a large-scale event notification infrastructure In *Proceedings of NGC*, London, U.K., Nov 2001.

[15] DABEK, F., KAASHOEK, M., F., KARGER, D., MORRIS, R. AND STOICA, I. Wide-area cooperative storage with CFS In *Proceedings of SOSP*, Banff, Canada, Oct 2001.

[16] HAND, S. AND ROSCOE, I. Mnemosyne: Peer-to-peer steganographic storage In *Proceedings of IPTPS*, Cambridge, CA, Mar 2002.

[17] ROWSTRON, A. AND DRUSCHEL, P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility In *Proceedings of SOSP*, Banff, Canada, Oct 2001.

[18] RHEA, S., EATON, P., GEELS, D., WEATHERSPOON, H., ZHAO, B., AND KUBIATOWICZ, J. Pond: The OceanStore prototype In *Proceedings of FAST*, San Francisco, CA, Apr 2003.

[19] ZHUANG, S., ZHAO, B., JOSEPH, A., KATZ, R., AND KUBIATOWICZ, J. Bayeux: An architecture

for scalable and fault-tolerant wide-area data dissemination In *Proceedings of NOSSDAV*, Port Jefferson, NY, June 2001.

[20] KARGER, D., LEHMAN, E., LEIGHTON, F., LEVINE, M., LEWIN,D., AND PANIGRAHY, R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, May 1997.

[21] http://www.kazaa.com