A Glance at Peer to Peer Systems

Omer Sinan Kaya

May 14, 2005

Contents

1 Introduction

Peer to Peer (P2P) systems are described as a network of computers that is formed by nodes (peers) connected through the Internet (overlay network) to achieve some common

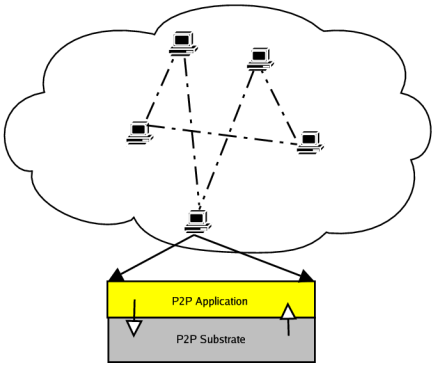

Figure 1: P2P Nodes

goal or to share resources among the computers for applications. [39] makes definition of P2P systems as follows:

"A Distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers). These shared resources are necessary to provide the Service and content offered by the network (e.g., file sharing or shared workspace for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requesters (Servant)."

Peers in P2P systems consist of two layers which are presented on Fig.1. On each node there is a middle-ware like software named "P2P substrate" which makes the hard work like connecting the nodes, routing, caching and other network maintenance works. This layer provides an API to the applications.

Most of the time, there exists confusion for the definition of P2P systems and Grid computing [41], [14]. Although these two systems closely resemble each other, they have some basic differences on their working environments. For example, grid computing systems comprise of powerful dedicated computers and CPU farms with high bandwidths; whereas, p2p systems consists of regular user computers with slow network connections. As a result, p2p systems suffer more failures than grid computing systems. Moreover, a grid computing environment usually has computers at the orders of thousands. However, p2p systems can reach up to millions or in theory to every computer in the Internet. Consequently, p2p systems are expected to be more scalable than grid systems.

Applications use the underlying services for their own problems. There are many kinds of applications. The most popular one is file sharing. Besides from document and file sharing, P2P services can be used for many other scientific purposes [3] or for brute force attacks again well known scientific problems such as gene decomposition. The technology fits into any place where huge numbers of CPU power, system resources and parallelism are required.

P2P concept is to set up a network of nodes without any centralized component and each node acts as either client/server or a forwarder for other nodes. Since every computer decides to join the network as a server, every user decides whether to dedicate some of its CPU power or network bandwidth by use of others. When many hosts in the Internet decide to join a network, a network of nodes with nearly 1M hosts is set up. For example

Kazaa [23] system has 60 million users in total and nearly 1-5 million users are on-line at anytime. With this large number of resources, some scalable and robust architectures are needed to function properly.

The system consists of heterogeneous architectures such as Linux, UNIX, Windows, Macintosh and different communication media such as Digital Subscribers Line (DSL), Cable Modem, Public Switched Telephone Network (PSTN) connections and so on. So the system must be able to handle different kinds of communication speeds as well as different architectures and operating systems. P2P services are useful for the Internet architecture, on the other hand by the introduction of wireless communication media and MANETs recently the system is also expected to handle mobility as well.

P2P systems are subject to frequent failures. Since a user that provides a file to the system can switch off his computer at any time, the system must be able to handle high failure rates. In order to achieve this, some caching techniques and replication schemes are applied. However, these solutions bring new questions such as when and where to cache data? Or when will the cache become stale? , How to invalidate a cache or should we trust the peer that invalidates our cache? Can it be a temporary network failure? How to construct coherency among node caches? What would be the best way for object replication? How many replicas do we need? ... These questions are the exciting part of P2P research.

One other problem is that a node might not want to share all of its resources and it always asks for its own benefit. For example, if a person is willing to download a music file, he or she wants to retrieve it as quickly as possible. However, if the user assigns the entire bandwidth for serving purposes then it will take more time to retrieve a file. So, system designers must keep in mind that different users will ask for different levels of resource sharing parameters. Some p2p systems such as [25] introduced this property by issuing an α parameter. This parameter is chosen by the user and quality of the search depends on this parameter.

2 First Ramp Up

The first initiative of P2P networks was Napster which was developed in 1999 by Shawn Fanning who was a BS student at Northeastern University. The system became so quickly widespread that it attracted some attention from the music companies and in the end it had to be closed.

Closing Napster at that time was a solution but the idea remained for future scientific studies. Many other derivatives of the protocol have been studies by many groups at universities and many of these projects were kept open-source for others to improve the system.

Some mostly cited studies include Kazaa [23], Gnutella [1], Morpheus [28], CAN [35], Chord [40], Pastry [38] and Tapestry [45].

3 P2P Architectures

Peer-to-Peer systems can be categorized as centralized, hybrid and distributed. Being the first starter, Napster was designed as a simple centralized system, in which records of the existing files and locations of the files were stored in a central server. Users would query

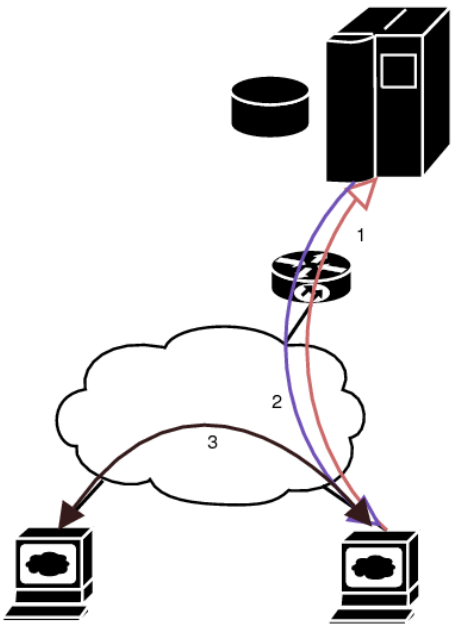

Figure 2: Napster Architecture.

the server for finding the location of the files and contact the node directly for further data exchange

3.1 Centralized Approach

A general view of the Napster architecture is given in Fig. 2. There is a central resource rich server connected to the Internet backbone via a high bandwidth network infrastructure and user computers query Napster server for files and as soon as they get the results they contact the computer that has the file.

Although Napster worked quite well in its times, it was discovered that there is a need of decentralization of functions. Since the fail of a single server might collapse the whole network or as the number of users increases the network load on a single point would increase tremendously, it is needed.

3.2 Hybrid Approach

Napster was followed by introduction of Kazaa [23] servers which remained popular for long duration until a Trojan horse has been detected in Kazaa clients. The purpose of this application had been to distribute some content for some companies and make money while providing the service for free. Another reason had been to collect user interests such as mostly visited web sites and sell it to content provider companies. A snapshot of the Kazaa client is given in Fig.3. From the figure, ad-ware can be easily seen in the box below.

The inclusion of spy-ware has also introduced the question of security in Peer-to-Peer services. Several variants of Kazaa application, like Kazaa-lite, that claimed not to include any spy-ware were released later on. But as the faith of Napster, Kazaa also had some problems with music companies and with the newer versions of Kazaa software,

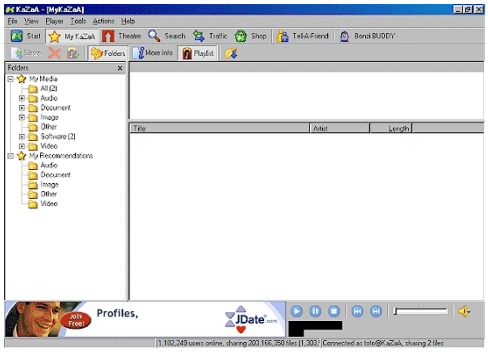

Figure 3: Kazaa Screen-shot.

they included software that reports user downloads or uploads to an organization. The idea has been to fine Kazaa for mostly copied files and music files.

Kazaa system consists of a decentralized and a centralized part. Nodes in Kazaa network are classified as Super-nodes and nodes. Super-nodes are assumed to have more resources such as CPU and bandwidth and they act as a server for clients. They index the available files on the network and queries are made from these super-nodes. This sounds logical but there are certain problems that need to be tackled such as how to find which nodes are super-nodes and how to keep synchronization between super-nodes.

What exactly is done with Kazaa is as follows: There is a central server that keeps track of user registrations and this server also counts the number of uploads and downloads. When a user wants to join the network, it first contacts the central server with a user name and password. Afterwards, the server provides the client with a list of super-nodes existing in the network. Meanwhile, clients find themselves a suitable super-node for connection using response times and make a connection to the selected super-node. From this point, super-node acts as a server for the client. In case the information that client queried does not exist, super-nodes contact other super-nodes to seek an answer. A similar scenario is given in Fig.4.

3.3 Distributed Approach

In P2P distributed systems, there exists no single point of failure. All nodes are assumed to be functionally equal. Because there is no centralized component, distributed algorithms for service discovery, neighbor tracking, location finding and message routing are required. P2P distributed systems are classified as structured and unstructured. Generally speaking, in unstructured p2p systems files can be stored on any node regardless of the structure of the overlay network.

3.3.1 Unstructured P2P Distributed Systems

Gnutella is a fully distributed P2P system developed by creators of Winamp at Nullsoft which was later purchased by American Online (AOL). In 1999, developers at Nullsoft created this software with a hack in 14 days. This software was provided on the Nullsoft web site under GPL license for two weeks. Later on, AOL Company discovered the potential market outbreak of this application and it was removed from the web site. However, they were too late and this duration was enough for the Internet enthusiasts

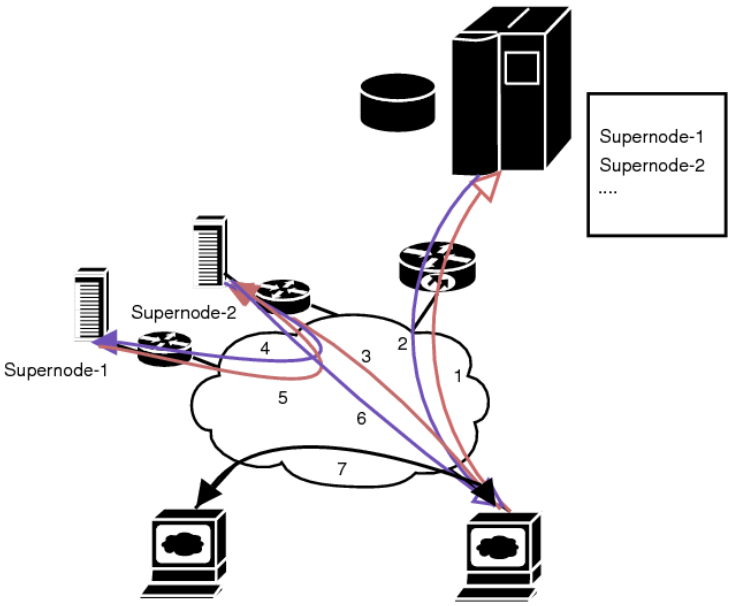

Figure 4: Kazaa Architecture.

download the software. The Gnutella protocol is well studied in the literature and it has been already reverse-engineered or many open-source or commercially available alternative clients, which support full Gnutella specs, are already out in the Internet. These software include Bearshare [4], Gnucleus [15], Morpheus [28], Limewire [24] and Phex [33].

There are five types of messages in Gnutella system:

- **Ping**: This message is used to discover nodes in the neighborhood and it is usually broadcast.
- **Pong**: When a peer node receives a ping message, it replies with a Pong message to indicate that it is present.
- **Query Messages**: Query messages are unique in the network and initiated with certain Time to Live (TTL) value to look for a document. These query messages are flooded in the direction of other hosts in the network. Some caching mechanism is used to improve the system to avoid transmitting messages that were already sent.
- **Query Response Messages**: Response messages follow the route of query messages to reach the initiator peer.
- **Get/Push Messages**: These messages are used for uploading or downloading messages. If the user is behind a firewall, push messages are used for downloading. Two users that are behind firewalls are not able to communicate with this system.

The downside of using an unstructured network comes from communication. In Gnutella, searches are made by flooding. Therefore, the cost of a search is too high. Although there have been some improvements proposed [44] by the research society, this system still suffers performance problems when compared to others. Another problem

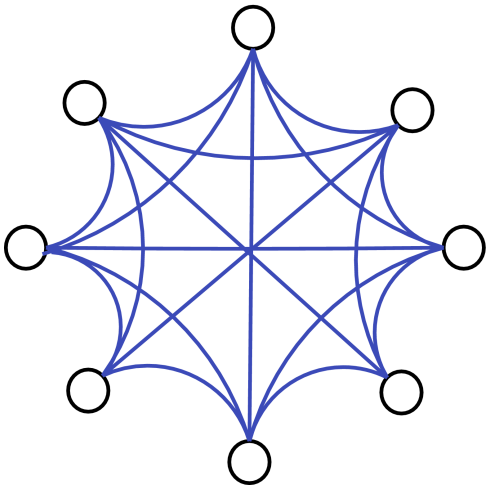

Figure 5: DHT Mesh Structure

with unstructured system is that it cannot give a guarantee for finding a file; whereas, a structured system will find a file if it exists on the network.

3.3.2 Structured P2P Distributed Systems

Most of the structured distributed peer to peer systems are using a hash namespace. Anyone, using these kinds of systems, calculates the hash value of a filename or a combination of a filename and a user name into a unique id that can be queried by each node. After that, nodes try to route data to nodes that have closer hash numbers for a given request. This way of looking up a value is often named as lookup using Distributed Hash Tables (DHT) in the literature. By comparison with the unstructured, hybrid and centralized systems, the general structure of DHT-based systems is a mesh network as shown in 5.

Most of the time, the hash function is either SHA-1 [12] or MD5 [37] which produces with high probability a unique value for a given input and they are completely one way. Consequently, having a similar hash value does not mean that the items are similar. This property makes these systems completely decentralized although many hop counts (some node might be located at totally distinct geographic places) are achieved before targeting the real host and this leads to poor design. However, some heuristic algorithms such as hop count or delay have been introduced by the systems to achieve better performance.

DHT-based systems make a node responsible for certain range of keys. The division of key address space varies between systems. Some organize into some shapes like rings, trees, hypercube and thorus. If the structure is a tree as in [17], the search is routed toward the parent node if no match is found and the parent node forwards the request to another child. This operation continues until a response is retrieved. A guarantee, which is usually logarithmic, is given that the final destination will finally be reached in several steps.

4 P2P Evolution

Another kind of classification is made according to the appearance of the P2P systems to the society. First generation P2P systems are Napster and Gnutella which are basically easy to implement and do not contain much optimization. 2nd generation P2P systems are more sophisticated and they often use DHT-based routing algorithms. Their purpose is given a query efficiently routing a message to the destination. They create a form of virtual topology and are generally named as structured P2P systems. 3rd generation P2P systems are also variants of structured P2P systems. However, they put more effort on security to close the gap between working implementations, security and anonymity.

5 State of The Art

5.1 Freenet

Freenet [8] is a structured P2P system. The motivation of Freenet is to protect anonymity of information and privacy. Specifically, Authors try to avoid eaves dropping on the communication, intended deletion of certain shares, monitoring and logging of user activities. The keys, named as Globally Unique Identifiers (GUID), are also derived using hash functions (SHA-1).

Network discovery is done by out-of-band means by broadcasting messages with certain hop limits to find a node connected to the network. Once connected, each node retrieves routing tables from the network.

Routing is done to find the closest match for a given ID as in other DHT-based systems. Freenet uses "Steepest Ascent Hill-Climbing Search" algorithm. A node will forward a request to a node that thinks it to be closer to the data item. If the chosen node cannot find the destination, it will return to the originator with a failed message. Next time, the node will try with some other node. To limit searches and resource usage, queries are tagged with certain TTL values. This means that any node, which is at the far end of the network, will not be reachable by Freenet peers. Once the connection is established, nodes with closer ID can cache the data to improve connectivity next time.

Most effort on this study has been put to security. Other structured P2P systems focus on efficient location of data items and Freenet extend these works with security in mind. As in other P2P systems there is a need for naming system. As most of other structured P2P systems look up files by taking hash of the file name and afterwards, Freenet does the same and routes requests toward nodes with similar hash values. However, there exist two kinds of GUIDs in Freenet. The first one is *Signed -Subspace Key* (SSK) and second one is *Content Hash Keys* (CHK).

SSK consists of two parts. First part is the public key of the file holder and second part is the name of the file. In this way, a hierarchical namespace is constructed. Hence, the hash value of the first part and second part is concatenated and used as search key. Upon creation of files, they are signed by the owner of the file and users check to see if the signature matches with the one they calculated. Usage of private/public key pairs guarantees updates to be done by only the owner of the file or private key holder.

CHK is the primary data-storage key which is computed by calculating the hash value of file content. As a result, every time a file is modified, it yields in another hash value. CHK keys can be provided on a web site and user can search with this key on Freenet but since CHK keys are binary they are not quite appropriate for user interaction. As a

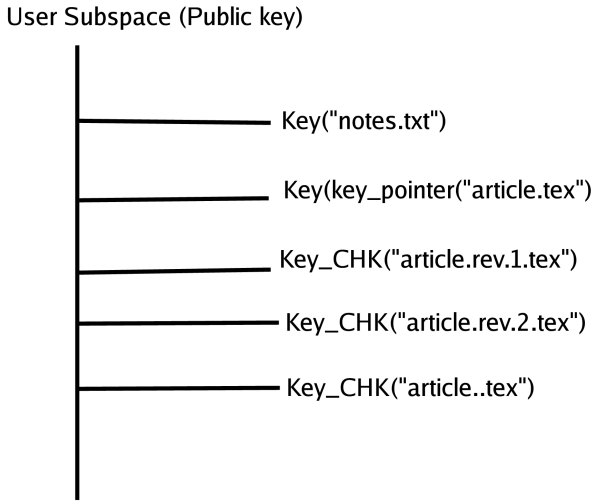

Figure 6: Freenet Naming System

result, authors use CHK in conjunction with SSK keys. An example of a namespace is given in Fig. 6. CHK keys will be stored in user subspace and there exists a pointer to the actual file is CHK in the subspace while maintaining the old ones. A user will make a search for the file name in user space. Later, he will retrieve the CHK key value in two steps. First, he will resolve the key value of the actual filename and use the actual filename is CHK key for search in Freenet. For users, who have downloaded older keys will still be able to access the old file if it is stored in the user repository since CHK is calculated over file contents. Others, who want to resolve older revision files, can still use the subspace.

Eaves dropping is avoided by encryption. Each node-to-node communication in Freenet is encrypted and a message traverses multiple hops to arrive a destination node and each node in the middle do not have any idea about the content of the file since it is encrypted and origin and the destination of the file since they are hash values. To achieve anonymity, each node in Freenet randomly lies about a document and claims itself as the owner. Consequently, it is nearly impossible to find the origin of a document.

Briefly, we can describe Freenet as P2P file storage and distribution system rather than a P2P file sharing system. Most popularly accessed files are replicated among nodes with similar IDs to achieve high availability and fault tolerance. Freenet protects user privacy at every level of the system (from communication to the local storage). Each user participates to the network by providing some disk space. This disk space is used for file replication and caching purposes. Least Recently Used (LRU) scheme is used for caching. Therefore, Freenet system will eventually delete files that are not accessed any more and it does not give any guarantee on finding a file. Since files are encrypted, a user does not know which file is stored on its file system. One problem with this system is that since each user joins to the network without taking any authorization, a malicious user can enter the network and use it for performing Denial of Service (DOS) attacks by filling user file space with junk data. Freenet relies on trained routing tables. If there is no or little node in the network or they are not yet connected, routing of a request may travel each node one by one until the destination is reached.

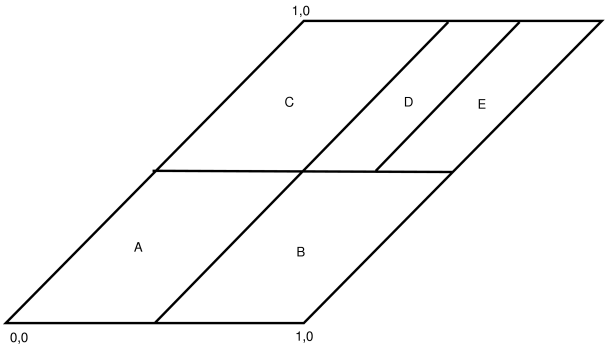

Figure 7: CAN Address Space

5.2 CAN

The Content Addressable Network (CAN) [36] is a distributed infrastructure that provides hash table-like functionality on Internet-like scales. In this work authors are seeking for Internet-scale naming system which is location-independent and fault-tolerant. CANs resemble a distributed hash table and operations include join, delete and lookup. Each node stores a chunk of the entire hash table.

The CAN design centers around a virtual d-dimensional Cartesian coordinate space on a d-thorus. Hash address space is partitioned using Cartesian coordinate space and nodes take responsibility of a certain zone in the coordinate space. An example partitioning of address space is given in Fig.7 where nodes A, B, C, D and E partition the address space and own the regions. Given a key, it will be mapped onto a coordinate value using hash function on the address space and it will be stored at the owner of the zone that P resides in. Retrieval is similarly quite trivial. A node looking for a key value will calculate the coordinate P and ask for the region owner to deliver the value to it.

Routing in CAN is based on coordinate space. Each node maintains the addresses of the owners of adjacent zones and constructs a coordinate routing table. A CAN message includes the destination coordinates. Using its neighbor coordinate set, a node routes a message toward its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. For a d dimensional space partitioned into n equal zones, the average routing path length is $(d/4)(n^(1/d))$ hops and individual nodes maintain $2d$ neighbors. This means that routing table size is independent of the number of nodes. A node will try to reach a destination using best candidate from the routing table. However; in case it fails, the routing algorithm will backtrack and choose another candidate. If connections to all neighbors are lost, Expanding Ring Search (ERS) algorithm will be used to discover any node that is connected to the network.

Virtual address space in CAN is dynamically partitioned into smaller peace. New node arrival starts with discovering a bootstrapping node which maintains a list of active the CAN nodes. DNS mechanisms are used to discover these nodes. Later, the node connects to the bootstrapping node and the joining node picks a random point in Cartesian space. Using the bootstrapping node, the joining node will contact the region owner of randomly chosen space and that region is partitioned into two for the new coming node. Keys that belong to the partitioned zone will be transferred to the incoming node. Additionally, update messages are sent to neighbors so that they do not have a stale routing table.

Each neighbor periodically sends heartbeat messages to each other to prevent network

partitioning in the case of failures. If a node does not reply within a certain time, it is assumed to be unreachable. Each node that detects the node to be unreachable will start a takeover process. Meanwhile, it will be listening for takeover messages from the surroundings. If any other node completes the takeover before the current node, takeover process is canceled. The takeover is done to recombine address space. To prevent stale routing information from being propagated in the network, each node that stores a key in the network is required to refresh its status periodically. If the node does not reply in a certain time, it is assumed that the key is not valid any more and will be deleted from the node.

The CAN performs better in terms of node states. Other systems usually have around $logN$ states per node whereas a CAN node only keeps $2d$ states. Nevertheless, the CAN also comes with some drawbacks and heuristic improvements for its own problems. The CAN routing algorithm is not as efficient as the others. To increase routing performance, authors propose to use Round Trip Timer (RTT)-weighted routing, replication of entries and increasing dimensions. Increasing dimensions comes with the cost of increased per node state; however, it greatly reduces application-level hop count because each node will have multiple neighbors that are connected to each other and contacting a next hop will take smaller paths, hence latency. Load balancing is introduced by including multiple nodes in the same region. These nodes actively communicate with each other and exchange keys between them. As a final word, some open questions in the CAN include security and anonymity.

5.3 Chord

Chord [40] is yet another distributed, scalable, structured P2P substrate. The main focus in this research is on efficient lookup of keys. As most of the other structured systems, well studied DHT-based approach is used in Chord. Cooperative File System (CFS) [9] is built on top of the Chord substrate. Authors use client and server terminology instead of substrate and application terms. CFS uses the basic functionality provided by Chord lookup system. Files are replicated in the system and efficient search algorithm of Chord is used to locate objects.

In this system, each node maintains routing information about $O(logN)$ other nodes and a lookup operation can be completed in $O(logN)$ steps where N is the number of nodes. Chord tries to improve system load via load balancing. When a new node joins the network, average of $O(1/N)$ keys are migrated to the new coming node. This update requires $O(log_2 N)$ message exchanges. They also propose certain heuristics for moving commonly accessed keys to resource rich servers. However, this property comes with the expense of maintaining state information in each node. Chord gives "best-effort persistence" guarantee for accessibility of an object. As long as there is one of the r nodes storing a key is available, Chord tries to make it accessible. In the case of network partitions, Chord adaptively organizes into smaller networks and provides service using this network. When the network partitioning is overcome, a stabilization protocol merges different hash or key domains into one.

Chord uses Consistent Hashing [22] to match object IDs with node IDs. Basically an object is stored on a node with bigger and closest ID, named successor ID. Suppose that the key consists of m bits. At each step, Chord tries to find the immediate successor for ID range $n+2^{(i-1)}$ and $n+2^{(i)}$, where $1 \leq i \leq m$. Consequently, during the search operation at each step node will halve the distance between the target and itself and if there exists

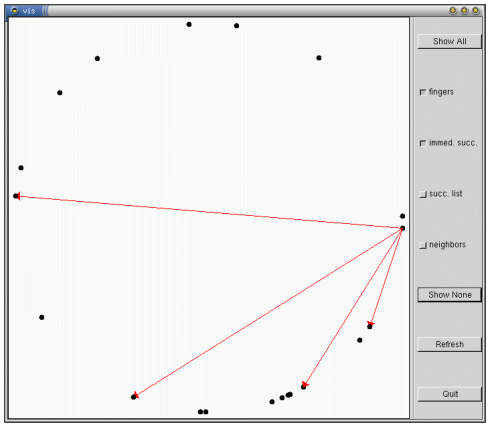

Figure 8: Chord Key Lookup

n node in the network, the algorithm will find the node at $O(logN)$ steps. Each node maintains a small number of entries in its routing table. If the node that holds the key is not reachable directly, the node will forward the request to the immediate predecessor of the range and ask that node to resolve the request. A sample run for 3 bit numbers IDs is given in Fig.8. Here; node 0 will allocate the regions from 1-2 as its first interval with successor node 1, interval 2-4 with successor node 3 and interval 4-0 with successor 0.

When a new node joins the network, it uses the information from the predecessor and successor of itself to initialize, store keys and configure pointers. Some of the keys that new node is responsible are moved to the newly joining node and pointers in the neighbors are updated. In the case of failures, nodes will lose their contact with the circle. To achieve this, authors propose to use a list of r successor neighbors. So that, any one of them can be contacted with a probability to remain on-line.

Another step for improving system performance is usage of local proximity metric for lookups. A node will try to find a key in its close vicinity before contacting other nodes. Closeness property is measured by round trip time. Each message transmitted also contains the location of the Node. Any node that decides to cache this entry will store the ID of the message sending node as well and it will be notified when the cached node fails.

Chord is a simple decentralized P2P system and it has been widely used in the research. Chord gives certain guarantee on finding an object. Either it is found or it definitely does not exist on the network. Chord study is focused on lookups and authors do not give any detail about anonymity and security issues. Problems with Chord include: link asymmetry and lack of in-place notifications. The number of hops traversed for accessing node A from node B might be different from traversing to node B from node A. Because the distribution of neighbors and number of entries in the routing table depends mostly on the location of the node in the circle, another problem is with node joining. A new coming node needs to contact its immediate successor which can be at the far end of a network to make its first initial connection. This is expensive in terms of maintaining consistency and network bandwidth.

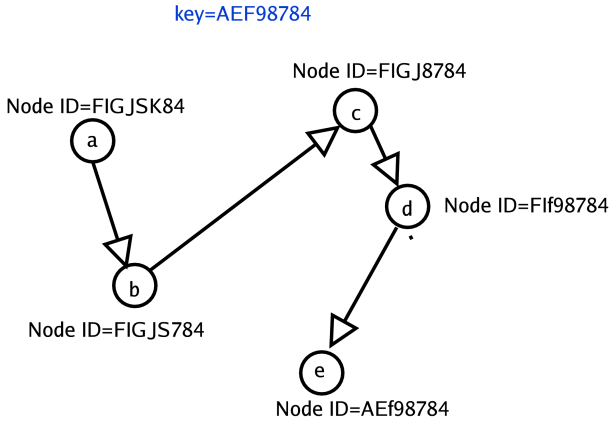

Figure 9: Pastry Routing

5.4 Pastry

Pastry is a distributed overlay network that is intended to be a routing layer for applications. Many applications have been developed on top of Pastry routing system such as Past [11], a file storage systems and Scribe, a publish/subscribe system [7].

Past is an application that is built on top of the Pastry routing algorithm. Past is a persistent, distributed storage system that uses smart-cards from brokers for disk space. It can be seen as an extension of the ideas on Freenet [8] where private/public keys are used to generate node IDs. Herein, the system is extended with a third party trusted dealer that creates private/public keys for the user and stores them in a smart-card with its certificate. When a user tries to contact another peer, its identity is checked by verifying signatures using certificates from the dealer. By this way, malicious users are not allowed to make use of the system. However, a malicious user can break into some smart-card software and try to exploit it. Assumption is that smart-cards are secure enough for not to be hacked. Peers exchange disk space by using the credit in their smart-cards. This credit is also charged for the user selected replication factor and file size. So that, a fair distribution of files is achieved.

Scribe also uses Pastry routing system. It uses hash of the topic names as key IDs. Subscribers send messages using these IDs. All the nodes on the ways cache these requests. Hence, toward the point where the topic is actually stored a multicast tree is constructed. A publisher also sends a message using this Hash value and all the nodes that have already cached subscribers (members of the tree) relay the message to the subscribers.

As other structured P2P systems, Pastry makes use of DHT and routes messages toward nodes with closer DHT values which can be located at distinct geographical places. Hence, the topology is much like virtual mesh topology. This gains Pastry the decentralization property. To avoid messages traversing distinct locations, Pastry extends routing tables with local node information. The Locality is achieved by taking into account the number of real network hops a message takes to reach a destination. So the lower the hop count, the closer a destination is. A node will check to see if there exists any local node to relay the message first.

Routing is done toward the node IDs with similar or bigger prefixes. Pastry can route a request in $log_{2^b} N$ steps under normal operation where b is the number of prefix bits

that are shared in the routing table.

Routing is done at each node toward nodes that have more common prefix with the key than the node has with the key. Fig 9 shows an example run for a Pastry routing system. At first step, node a will send the message to node b. Node b, which has more common prefix with the key, will forward the message to c and this will repeat until the final destination is reached.

The number of entries required at each node is $(2^b - 1) * \lceil log_{2^b} N \rceil + l$. A key in Pastry is formed as a sequence of numbers from base 2^b (b is most of the 4). There are $2^b - 1$ entries at each row of the routing table. Each entry at row n contains the same prefix with the other nodes at the same level for nth digit but different value for $n + 1$th digit. l is the number of nodes with similar node IDs.

Join operation is performed by out of band means using Expanding Ring Search (ERS) algorithm. To avoid network partitions, authors propose to do ERS regularly so that there exists always a node in the close vicinity that is connected to the overlay network. A node, which acts as a volunteer for forwarding node join operation, delivers this to the nodes with similar node IDs. When a connection to the node with similar ID is achieved, a routing table is downloaded from the node and a join notification message is sent to other nodes with close node IDs.

Pastry is a structured locality optimized P2P substrate. It provides API to the application layers and PAST and SCRIBE are examples of applications. According to the authors' results, Pastry is efficient in terms of routing table size with a common bit count of 4 and node number of 10^6, a routing table contains, on average, 75 entries and the expected number of hops to reach a destination is around 5. Files are replicated at different replicas according to a user given parameter named k which determines the quality of service expected from the system. Since the nodes contain state information of the network in their routing table, there is a need to keep consistency. Authors propose to exchange heartbeat messages between nodes in the close address space. If a node is detected not to respond for a certain time, all the nodes update their routing table. However, instant exchanging of update messages waste the network bandwidth.

5.5 Tornado

Tornado [17] is an early implementation of a P2P system with distributed hash tables and they further try their application with MANETs [18]. It uses, as most of the other P2P systems, distributed hash address space for specifying items.

There are three basic characteristics that make Tornado different from other schemes. These are:

- Virtual home concept
- Classification of nodes as good and bad
- Node autonomy.

Tornado introduces the definition of virtual home which assigns a data item to a virtual address instead of the real address of the node. By this way data items are stored at some other serve willing nodes instead of a real node. The address of the virtual home is selected to be numerically close to the actual item and in general items are routed toward virtual home addresses.

Tornado is a distributed system in the sense that each node decides which virtual homes will be its neighbors or not. Another feature that Tornado provides is the making a distinction between nodes as good and bad. Only good nodes are elected to be virtual neighbors and if it happens the case that a virtual neighbor is overloaded another neighbor is selected. Election of neighbors as good and bad neighbors are done by each node so the system is completely autonomous.

In order to cope with heterogeneous networks, Tornado classifies nodes as good and bad and even as active good and inactive goods. Nodes with higher resources are marked as good nodes and among the good ones, nodes with highly reliable network connection are highlighted as active-good nodes. An inactive good node will become active when an active node becomes overloaded. The capability of each node is retrieved from the existing system and different load conditions are executed on different platforms.

The lookup operation is performed as follows: Each node maintains a k-way tree-like structure and each item in the tree is responsible for the lookup of a range of hash values. So depending on the value user is interested, a specific node is selected from the tree and message is forwarded to that node. The tree construction is as follows. R is the size of the address space. A node will elect its uppermost level neighbor by finding a node that has hash values $R/2$ away from it. This is done by taking the difference between the node ID and the target node ID. Later it will continue with $R/4$ and $R/8$ and so on for k=2. Once this tree is constructed, messages will be transmitted to relevant leaders.

If a node wants to join the network, it starts the process by computing its own ID using a hash function. Later on, it finds a node connected to the network using out of band messages. Once this node receives joining message it forwards the message to a node with similar ID and an entry named as virtual home entry is created at this node. When the node is registered with virtual home, other upper nodes in the tree are contacted to update their routing tables. Then, the joining node is contacted again with the updated routing table. From now on, this node can also act as a virtual home for other nodes because it has an identical routing table. Nodes that receive and forward the messages can cache this information for further performance improvements.

Discovery of a nearby neighbor connected to the network is achieved by local broadcasts. Once the connection is set up, each node periodically refreshes its membership information so that routing tables are kept updated.

The main idea with virtual home concept is to decrease the number of hops traversed by a query message. A good node in the network takes the responsibility to react to the query for this hosted node. This greatly decreases latency and the number of hops and communication is kept local between resource rich devices.

The publishing of a data item is also done in a similar way. The node that wants to publish certain data queries the virtual home with closer node ID to check if there is enough storage. If the answer is yes, the data item is stored at the publisher node. If not, publishing is aborted. Once the data is published, good nodes with similar IDs can replicate this object for further data availability. According to the simulation results they can still achieve nearly 100% availability at 20% node failures with replicas numbers over 4.

Another feature that Tornado provides is to access geographically closer nodes for network connection additionally. Forwarding is done to a leader that is closer during a querying phase. Closer definition is made by measuring time delays between nodes and making use of network positioning systems such as Global Network Positioning (GNP) [31] which provides coordinate information for a node in the network. They test their

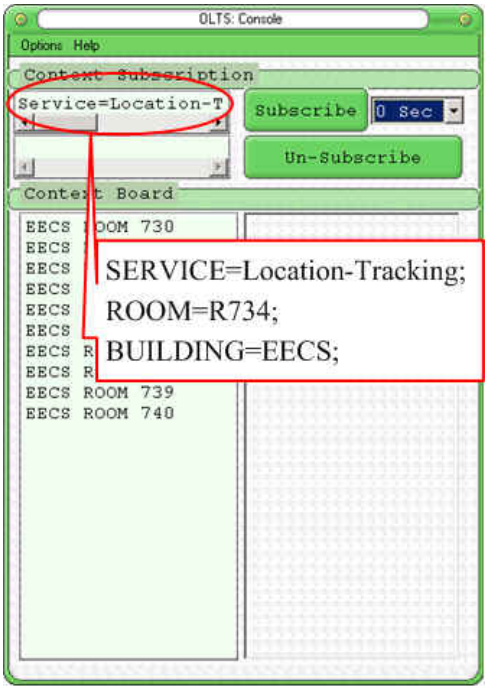

Figure 10: Trailblazer Screen-shot.

simulators using their own built simulator (another own built simulator). According to the results, the usage of local information for construction of newly joining node decreases the number of messages by 23% percent. In this scheme, there is no volunteer node that asks for information but a newly joining node collects its routing information from the neighbors and decides itself. In the case of connection loss, they propose to use redundant paths between leaders to increase connectivity.

They have implemented two applications: a virtual music scoring system and an object-location tracking system using this P2P system.

Future work includes testing the system under heavy mobile environments and trying to make more use of location information.

5.6 Kademlia

Kademlia is a peer-to-peer storage and lookup system. Files are looked up using 160 bit key values which are generated by hashing the name of the file. Kademlia tries to route messages efficiently in the network to reach a destination node. Kademlia is different from other researches in the context that they try to minimize the number of control messages exchanged and build a fault tolerant system. They embed this information into the key lookup information. Parallel asynchronous Remote Procedure Call (RPC) messages are used by this system to avoid delay mechanisms in the network and to quickly retrieve a key initiating parallel search in the network.

They also give some important figures about node behaviors in real network deployment. One of the most important results derived from their work is that a randomly selected node will stay on-line with a probability of 1/2 and nodes that remain on-line for longer durations tend to remain on-line for more. So, one should keep in mind the fact that older routes in the routing tables are more reliable. Another observation is that

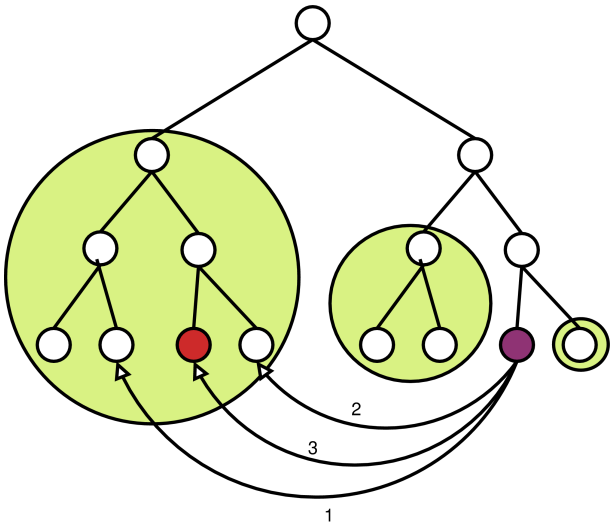

Figure 11: Kademlia Binary Tree

a randomly selected neighbor which is on-line has the probability of staying on-line for one more hour by 1/2. This means that frequent node join or leave operations exist in the network. Designed system should be able to tolerate this behavior.

They take similar approach for key generation and data storage in the network. Each node is assigned a unique ID which is usually hash of the IP address. Data to be stored is also assigned an ID and data is stored on nodes with similar ID values and when a key look up is initiated, search is propagated toward nodes with close ID values. Each node keeps track of $O(logN)$ neighbors and a lookup operation takes nearly $log(N)$ steps where N is the number of nodes in the network.

Kademlia uses XOR metric for finding the distance between the keys and uniqueness of Kademlia comes from this property. XOR is symmetric so that a lookup operation from any node will take the same number of steps to reach destination. Given a node Id and a key value, Kademlia defines the distance between them as $d(x, y) = x \oplus y$. Kademlia gains its symmetric property from the property of XOR as $d(x, y) = d(y, x)$. This is a weak property of Chord and authors try to improve this property. Kademlia treats nodes as leaves in a binary tree. The position of each node is determined by shortest unique prefix of that node. For any given node, the binary tree is divided into a series of successively lower *subtrees* that do not contain the node. An example tree is depicted in Fig.11. Subtrees contain one, two and four nodes respectively. The closer the subtrees in the tree, the less the distance between them are. Kademlia protocol guarantees that each node knows at least one node in other subtrees. Lookup operation is performed as follows. A node will first find another node in the same region and contact it for the target node. This node will return the ID of a node that it thinks is closer then itself. This time, the node will contact the new neighbor and it will also return another node with closer ID and this operation will repeat itself iteratively until the destination node is reached. One problem with this approach is that communication between a node and a closer node is performed iteratively from originating node. This communication can traverse longer paths because each node can be located at totally different geographical regions and resolving of a query requires a step by step query of these nodes. To overcome this problem, parallel asynchronous RPC is used by Kademlia.

Each node keeps track of k (system wide parameter) node IDs between 2^i and $2^i + 1$ where $0 \leq i < 160$ and 160 is the number of bits a hash consists of. These lists are called buckets by the authors and they try to keep old routes more on the list and discard new ones if the list is full. When the list is full and a new ID is received, the node with least recently used is contacted to check if it exists in the network and if not it is replaced with the new key. However, contacting the same node every time a new ID is received consumes lots of bandwidth. One optimization implemented by the authors is to keep status checking until a valid data item to relay to the node arrives. Meanwhile, newly coming ID is kept in a cache and IDs with most recently seen ones take the first order in the cache.

Most of the operations in Kademlia are done by using parallel lookup operation. If a node wants to publish some data, it retrieves k nodes with close node IDs and stores the data in these nodes. To keep the system intact, every data publisher is requested to update their data every 24 hours. Caching techniques are employed to speed up lookup processes.

A node that wants to join the network looks up itself using an already on-line node. After locating itself in the tree, it will send join notification to other nodes in that subtree which costs $O(log(N))$ messages. Nodes on other subtrees will update their routing tables as Request for Comments (RPC) are received. Cost of leaving in this system is 0. Authors use their observations for node connectivity. At each hour, routing tables are refreshed. Therefore, if a node leaves the network Kademlia will eventually reach to consistent state after some time.

Kademlia is a novel P2P system that provides symmetric lookup latencies. Questions like anonymity and security remain open in Kademlia. Each publisher needs to update its existence at some intervals. This may easily reveal user identity. A concurrency parameter α is provided to applications for trading better network performance. Nodes with higher α values will execute more number of parallel lookup operations. Hence, they will collect the result in shorter time.

5.7 Instant Messaging

Instant Messaging (IM) is another and most popular kind of peer to peer systems although user community is not very much aware of that. IM systems can be centralized or hybrid as in Yahoo Messenger [26], ICQ [19] and Microsoft Messenger (MSN) [29]. For instance, users of Yahoo Messenger register themselves at yahoo servers with a user name and password. Later on, they add their contact names to their user list and send message to each other using these registered user names. Once the registration is completed, users send each other messages through direct IP connection if necessary. From this point of view, this architecture resembles Napster servers where files (here it is contact names) are stored at servers and lookup operation is done using server database. (Again we use servers to look up for other users) Some systems like ICQ and Yahoo provide the user with the capability to store messages on the server if the user is offline and these messages are delivered when the user becomes on-line next time. From distributed point of view, this feature adds significant load on servers since a server might need to store messages for millions of users at any time to be delivered. Hence, not only is the storage of these messages a problem, but also delivery of these messages generates bursts of network traffic.

Other systems such as Jabber [20], an open source instant messaging system, try to

decentralize server capability. The architecture resembles email systems in which many email servers exist in the Internet and messages are routed through mail servers until it is delivered to the last user. In Jabber, there are many public servers whose addresses are published on the web site. Users register themselves to one of these servers or even may decide to set up their own server. User names in this system are in the form of username@server. So, before delivering a message a lookup operation for the server is made and message is delivered to the server where it is delivered to last user. One of the main concerns in Jabber system is to be able to exchange messages in different kinds of applications. Therefore, the communication messages are all based on Extensible Markup Language [43]. XML is a new Internet standard that allows arbitrary number of applications to exchange data in an interoperable manner. For those legacy systems such as MSN, ICQ and Yahoo Messenger, there exists on the network gateway servers which act as a proxy between Jabber and other systems. Users address their messages to these gateway servers and they deliver it to the target network and vice versa.

5.8 Jxta

Jxta [2] is an open source P2P architecture that is supported by many experts around the world. It focuses on knowledge management, content sharing and collaborative application markets and it grows to be a future standard on these topics. Some of the key requirements in this project are to achieve CPU, operating system, communication protocol and programming language independence. As a result, the use of technology varies from ad hoc networks on portable devices and sensor networks to desktop computers. This ad hoc nature of nodes constitutes one big difference from other system in design criteria. Most of the other P2P systems are designed for PCs that are static and do not change their location. When the environment is quite harsh and energy is the most demanding resource, then the existing protocols that depend more on frequent message exchange will no longer work or deplete the battery immediately without ever being used by the users.

Jxta architecture looks like Kazaa architecture. In Jxta, there is the notion of super-peers and peers. One difference in this system is that any node can decide to be super-peer if none exists within the network vicinity. Super-peer nodes act as rendezvous points in the networks. Clients advertise their services with certain timeouts to super-peer nodes with an index number. The uniqueness of these index numbers is achieved by Shared Resource Distributed Index (SRDI) service. Each super-peer keeps track of peer IDs and their advertisements and they send heartbeat messages to clients to keep their databases consistent. Each rendezvous or super-peer node also exchange message between them to exchange advertisements indexes. There exists a loose consistency between rendezvous points to save energy and indexes stored can be temporarily or permanently wrong.

Jxta uses the same concept of DHT lookup operation in the network but is extended with mobility in mind. When certain node cannot be contacted, other nodes that have similar IDs (+- 1) are contacted. Each node is also stored with seeding rendezvous points. If no contact is achieved within certain timeout, a node will try to connect one of those seeding rendezvous points until a connection to the network is established.

For those nodes, which are behind firewalls or Network Address Translation (NAT) devices; relay-peers take the responsibility to route messages to nodes inside NAT network. A node inside the network established a connection to the relay-peer. Later on, all traffic that is destined to node behind firewall passes through the relay node.

JXTA uses an adaptive source routing algorithm for locating nodes. Routes are advertised in the network and each route includes the exact path that it should follow to reach final destination. Nodes cache this routing information and as they see shortcuts in the network, they update their routing table with the shortcut to reach final destination in less hop counts. Transmitted messages also embody route information. This helps a receiving node to check for its freshness of routing information and use the incoming path for sending responses.

Pipes are used for making the connection between peers and these connections are advertised in the network so that any other node who wishes to contact the peer at the end of a pipe can use existing pipe for data transfer. Advertisements are again done by system wide unique IDs. All the communication in JXTA is secured using Transport Layer Security (TLS) connections. Once the connection is established, multiple pipes can be created on top of TLS connection. RSA with 3DES and SHA-1 combination is used for security purposes.

Jxta is an open source effort for standardization and is instantly being updated by the developers to meet user requirements. One of the biggest problems on JXTA that can adversely affect system scalability is achievement of system wide unique index values. On moving or very dynamic environments this is not possible. Authors try to compensate this weakness by assuming an optimal concurrency model and even the system can carry some wrong references. One other difference with other system is that resources are advertised in JXTA; whereas, at other systems users need to know the name of the resource that they are looking for. JXTA is a more complete application space for developers and contains a real implementation. Other researchers focus on a single subject and test their algorithms using simulators.

5.9 Free Haven

The Free Haven Project [10] is a design for a system of anonymous storage which resists the attempts of powerful adversaries to find or destroy stored data. Authors try to hid origin of the document while constructing a system resilient against security attacks. To provide anonymity, they use re-mailers and mix-nets. Each document is Free Haven lasts for certain duration. After this time, documents are automatically purged from the system. Files in this system are stored in pieces at different locations and trading system is used between peers to store a file. Each peer that promises to store a file is monitored by another peer. This system is called the "buddy system". Buddy system increases chance of location of files. Buddy system imposes a significant overhead in the network. In Free Haven, servers encrypt and send a share of a document after receiving a request. Requests can also be signed by other servers to check if the requesters are authentic. Files are encrypted during communication in the network. Nevertheless, they are stored on hosts as clear texts. This allows any malicious user to edit file contents.

5.10 Mojo Nation

Mojo Nation [30] is a peer-driven content distribution technology built around three core technologies: swarm distribution, decentralized services, and market-based distributed load balancing. Mojo Nation is a P2P system and in this system there exists no certain server. Each peer application contains an agent that provides network functionality for the user. Mojo Nation tries to distribute tasks among peers. There by, collaboration of

nodes is targeted instead of a single serving node and replication is achieved. Each sharing activity and peer interaction in Mojo Nation is related to credits named "Mojo". The purpose of this scheme is to provide better guarantees for users who have more ranking. If there are multiple requests for a file in the network, a user with higher ranking will move to the front of the request processing queue or will be directed to some other node that has lower workload. Some third party trusted organization is relied on for checking certificates and credit negotiations. For increasing performance, multiple agent searches are initiated for chunks of files. Mojo Nation is a P2P environment for digital distribution and consumption of content. Each user request in the system is charged with Mojo credits and user service depends on the amount of money paid. File chunks are stored encrypted in the system and they are reassembled and decrypted before delivery to the last user.

6 Conclusion

6.1 State of Research

The years between 1999 and 2001 have been the top point for P2P services in terms of usage. Since then, as P2P services received widespread usage, most people gathered ideas like which system is better and which drawbacks these systems have. It has been realized that recently no novel P2P system has been proposed and the research is more mature now. Research efforts are put on basic system constructs instead of designing a system from scratch. Most of the papers [44], [32] published usually makes some small improvements over existing systems. In contribution to this, most people have some doubts about using P2P services since P2P services introduced some security problems. As can be seen from the studies, there exist several basic research areas targeted by different architectures. The main focus is generally on security, routing service, location service, and fault tolerance and each system comes with one trade off against other property.

6.2 Architectural Issues

- Unstructured systems are easy to implement, but do not scale well.
- Searches in DHT-based systems are made for a single key. Hence, the probability of finding a file depends on stating exactly the same name prior to search. However, most of the users generally search for keywords instead of full file names and when using DHT-based the search will simply fail. Therefore, there is a need to augment DHT-based systems with partial filename searches.
- Unstructured P2P systems have some advantages and disadvantages when compared with others. For example, advantages include being more resilient against failures. Since the location of the file is not related to nodes, any node can fail independently and there is usually minimal overhead to maintain a network when a node fails. However, these systems generally do not give any guarantee about finding a file and searches are usually limited with certain hops. Techniques like Round Trip Time (RTT) / parallel search are employed to decrease the number of hops.

6.3 Performance Metrics

A survey paper [5], categorizes performance metrics as follows:

- Security
- Anonymity
- Scalability
- Resiliency
- Query Efficiency

When comparing a study with the other, we must consider how that system handles security, can it achieve anonymity, how scalable that system is, can it work in the case of failures and how many of the queries can be successfully resolved. Query Efficiency is closely related to location of the objects in the network. Resiliency is mostly tacked with replication and scalability is achieved by keeping the system as decentralized as possible. Routing algorithm directly affects query efficiency. Routing algorithm must ensure that a message travels minimum number of hops to reach target, the size of the routing table is small (distributed routing tables) and system is robust against node arrivals and joins.

6.4 Future

In [1] authors try to look at user is social behaviors in using P2P systems. They have collected some data from certain Internet Service Providers (ISP) and conducted some statistical analysis on gathered data. Authors reveal that nearly 70% of the Gnutella users do not share files and they try to increase their benefits from the system by either providing fake or useless data. Hence, file searches on Gnutella network is done from 1% of the total number of computers and authors convey their concern on the system being converted to a client/server based systems. This behavior leads to performance decrease in the system. Since any node, that does not share a file acts a intermediary gateway on the network, nodes that have actual files become more and more far away from the users. Because searches are made by certain TTL values, it is probable that these serving peers will not be able to be accessed any more. Another problem is that, as only small portion of the system serves for the network, there is a risk that these users can be identified and sued by music companies or these computers become so heavily loaded that other users will not be served. Consequently, to protect P2P system concept from transforming to client server architectures there is a need to force users to share before getting some data from the network as in the days of Bulletin Board Systems (BBS). One other way which is implemented by [8] is to replicate files on intermediary nodes without informing the user. This approach works well for most accessed files but it can also store some malicious files on user hard drive because user does not know which file is stored on his or her file system.

It is been reported [6] recently that P2P traffic is declining due to the pressure from Record Industry Association of America (RIAA). People at IP Monitoring [34] project provide a web site for collection of the Internet traffic for analysis. These analysis include packet and application level classifications as well as delay, loss and performance characteristics. They collect data from 60 monitoring hot-spots. Recent application breakdown

(6th February 2004) analysis of traffic shows that 25.59% of the traffic is occupied by file sharing applications, 26.55% of the traffic is Web traffic and 33.64% is other Transport Control Protocol (TCP) traffic. This other TCP traffic portion of network traffic may be generated by some other P2P application as well.

Another study [16] for monitoring network activity reports that 43% of the TCP traffic is occupied by P2P traffic while web traffic occupies only 14% of TCP. Measurements include applications like Gnutella, Kazaa and Akamai [13] applications. Results show that P2P traffic dominates web traffic and it peeks to top point during the night. Authors also inspect the traffic content and it has been discovered that P2P applications are mostly used for transferring video. This implies user interest on video which can be a killer application for future handheld devices. Measurements also point out that Gnutella is the top bandwidth consuming application. Authors propose that there is a large potential for caching in P2P systems. However, it is been also argued that it takes months for a cache to warm up. The reason for this much delay is that, huge number of bytes needs to be transferred to caching point. After training, system works better.

In the paper [21], they try to find out if the usage ratio of P2P systems is declining as suggested by others. It is been argued that either measurements are not realistic or methodology is wrong. Moreover, new generation P2P networks use Hypertext Transport Protocol (HTTP) as transport protocol which means that peer to peer traffic is indistinguishable from web browsing traffic. In this paper, a new heuristic algorithm is proposed by the authors to distinguish P2P traffic from web browsing. Their have measured network traffic at an ISP for 2 years for testing purposes. They try to analyze packet contents to match for certain protocol. In order to analyze packet transmission authors reverse-engineered some existing P2P protocols. This study shows that P2P traffic is not declining but becomes difficult to inspect. The methodology used by authors uses pattern matching in packet contents. Some P2P systems such as Freenet use encryption. In this case, it is not possible to inspect traffic.

To summarize, it seems that P2P systems can really achieve anonymity and they have succeeded in accomplishing their objective. It becomes more difficult to identify sophisticated P2P from other traffic. P2P systems are now part of user daily life and no other novel systems are being announced recently. It is been expected that as Mobile Ad Hoc Networks become more widespread and if ubiquitous computing devised by Mark Weiser [42] becomes reality, P2P is expected to be the killer application. Peer to peer systems are yet ready for distributed, fault tolerant and secure data storage. Any user with a Personal Digital Assistant (PDA) or some device in their hand will be able to access their files or request some music from some provider. Combined with ubiquitous computing and mobility requirements many features of P2P systems need to be redesigned.

Another open problem is with standardization. JXTA is an open source attempt initiated by Sun Microsystems to establish an open source peer to peer application development environment. There exists a similar attempt from Microsoft under the name of .NET [27]. .NET environment is specifically supported by Microsoft operating systems and targets for web content publishing and access from mobile devices. It seems there is going to be strong standardization war between these two systems but as the systems standardize they become more specialized in meeting developer requirements.

References

[1] E. Adar and B. Huberman. Free riding on gnutella, 2000.

[2] Bernard Traversat Ahkil. Project jxta 2.0 super-peer virtual network.

[3] 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.

[4] Bearshare. http://www.bearshare.com/.

[5] Chonggang Wang Bo. Peer-to-peer overlay networks: A survey.

[6] John Borland. Riaa threat may be slowing file swapping.

[7] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 2002. To appear.

[8] Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[9] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[10] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[11] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. pages 75–80, 2001.

[12] D. Eastlake and P. Jones. Us secure hash algorithm 1 (sha1), 2001.

[13] F. Craig Floro, Rosemary Nelson, and Victoria Garshnek. An overview of the akamai telemedicine project: A pacific perspective. In *HICSS*, 1999.

[14] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–??, 2001.

[15] Gnucleus. http://www.gnucleus.com/.

[16] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 314–329. ACM Press, 2003.

[17] Hung-Chang Hsiao and Chung-Ta King. Tornado: a capability-aware peer-to-peer storage overlay. *J. Parallel Distrib. Comput.*, 64(6):747–758, 2004.

[18] Hung-Chang Hsiao, Chung-Ta King, Chia-Hsing Hou, Chuan-Mao Lin, and Wen-Hung Sun. Practical data-centric routing for large-scale wireless ad hoc networks. Technical report, Department of Computer Science National Tsing-Hua University, July 2003.

[19] ICQ. http://www.icq.com/.

[20] Jabber. http://www.jabber.org/.

[21] T. Karagiannis, A. Broido, N. Brownlee, K.C. Claffy, and M Faloutsos. Is p2p dying or just hiding? In *Globecom 2004 Conference*, 2004.

[22] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[23] Kazaa. http://www.kazaa.com.

[24] Limewire. http://www.limewire.com/.

[25] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric, 2002.

[26] Yahoo Messenger. http://www.yahoo.com/.

[27] Microsoft Corporations. *Microsoft .NET*, 2001. URL: http://www.microsoft.com/net.

[28] Morpheus. http://www.morpheus.com/.

[29] MSN. http://www.msn.com/.

[30] Mojo Nation. Mojo nation technology overview.

[31] T. S. Eugene Ng and Hui Zhang. Global network positioning: A new approach to network distance prediction.

[32] Multi-Tier Capability-Aware Overlay. Improving peer to peer search with.

[33] Phex. http://phex.kouk.de/.

[34] IP Monitoring Project. Ip monitoring project.

[35] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[36] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[37] R. Rivest. The md5 message-digest algorithm, 1992.

[38] 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.

[39] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing*, pages 101–102, 2001.

[40] 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.

[41] Douglas Thain and Miron Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[42] M. Weiser. Ubiquitous computing. *Computer*, 26(10):71–72, 1993.

[43] XML. http://www.w3.org/xml/.

[44] 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.

[45] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.