Searching in P2P Networks: A Survey

Bui Minh Nhat
Helsinki University of Technology
nbui@cc.hut.fi

Abstract

The communication environments of the Internet today are more complex than the traditional distributed systems. These environments are the inter-connecting between various types of network schemes, each of which has its own features and complexity. Among these network schemes, the Peer-to-Peer (P2P) network overlays not only gain much interest from researchers but from large amount of users as well. From the researchers' view, they provide a foundation for creating large-scale data sharing, content distribution and application-level multicast applications. And for the users, they provide environments for sharing content files (containing audio, video, data or anything in digital format) and real-time data (such as telephony traffic). In this paper, we will study five P2P network schemes (CAN, Chord, Tapestry, Freenet, Gnutella) by looking at their outstanding features and use cases. Furthermore, we will attempt to use the taxonomy to make comparisons between these schemes. This comparison helps to select proper P2P network schemes for specific applications.

KEYWORDS: P2P Searching, DHT, CAN, Chord, Tapestry, Freenet, Gnutella, Structured, Unstructured.

1 Introduction

Just like in many similar cases, the beauty of Peer-to-Peer relies on a pretty sophisticated technology running behind the scenes. It is becoming popular since the emergence of file sharing systems such as Bittorrent[2] and Napster[4]. Today, many operators realize more benefits of P2P and they motivate the development of products based on their needs. File sharing, content distribution, Internet telephony and video conferencing are taking advantages of P2P network architectures.

P2P networks can be classified into *Structured* and *Unstructured* overlays. *Unstructured* overlay networks are a key component in many P2P systems. The networks use flooding as the mechanism to send queries. Although flooding technique is poorly suitable for locating rare data, it is effective for locating replicated data and resilient to node failure or message losses. In *Structured* P2P overlay network, the topology is tightly controlled and content are placed at specific location that will make the queries more efficient. Such Structured P2P systems use the *Distributed Hash Table* (DHT) to achieve the goal. DHT-based system can locate data in a small *O(logN)* overlay hops on average, where *N* is the number of peers in the system [28]. DHT-based systems are an important class of P2P routing infrastructures. They support rapid development of many applications and enable scalable, wide-area retrieval of shared information.

When operators start a P2P service, they hesitate to choose among various network schemes. Each P2P network scheme has its own advantages and disadvantages. In this paper's scope, we will compare the features of five schemes: CAN, Chord, Tapestry, Freenet, Gnutella. Based on this comparison, the operators can realize which P2P network schemes are suitable for their services. The criteria for comparison are:

- *Decentralization* - evaluate the distribution of the overlay system.
- *Operation Architecture* - the architecture for the overlay system's operation.
- *Retrieve Protocol* - the protocol used for lookup process.
- *System Parameters* - the parameters for the overlay system operation.
- *Routing Performance* - the lookup routing protocol performance.
- *Routing State* - the routing state of overlay system.
- *Join/Leave Behavior* - the behaviors of peers when they join or leave the overlay system.
- *Security* - describe the vulnerabilities of overlay system.
- *Reliability* - examine how robust the overlay system when failures occur.

In Section 2 and 3 of the paper, we will describe key features of Structured and Unstructured P2P overlay networks and their basic operation functionalities. We will introduce 3 overlay schemes of Structured P2P networks (Content Addressable Network - CAN, Chord and Tapestry) and 2 overlay schemes of Unstructured networks (Freenet and Gnutella). After providing basic understanding of these overlay schemes, we will also evaluate and compare the features and performance of each scheme. In Section 4 , we discuss a proposed hybrid searching techniques in the P2P overlay networks. Finally, in Section 5, we conclude the paper with thoughts on some directions for the future in P2P overlay networking.

2 Structured P2P Overlay Networks

In the structured overlay network, the keys are assigned to data items and each peers are mapped into a graph that maps each data key to a peer. This structured graph allows an efficient data discovery using the given keys. However, this type of network does not support complex queries and it needs to store a pointer to each data object at the peer that is responsible for the key. In this section, we study and compare three Structured P2P overlay networks: Content Addressable Network (CAN) [25], Chord [18], Tapestry [14] [15].

2.1 Content Addressable Network (CAN)

In some P2P networks, a central server is used to store the index of all the files available within the peer community. If a peer requests a file, the server processes the query to obtain the IP addresses of peers storing the requested file. This peer-to-peer communication model is still much centralized. Further efforts are made to improve this situation by using flooding technique. But it is still unscalable as the size of the community grows. A new scalable indexing mechanism is introduced, it is Content Addressable Network (CAN) [25].

CAN is a distributed infrastructure that provides hash table-like functionality on Internet-like scale. The characteristics of CAN are scalable, fault-tolerant and completely self-organizing. As previously mentioned, CAN resembles a hash table. The basic operations in CAN are insertion, lookup and deletion the *(key, value)* pairs. Each peer stores a chunk (or a zone) of the entire hash table which holds information of a numbers of adjacent zones in the table. Requests for keys are routed by the intermediate peers to zones that contain the key. Using this technique, CAN uses no form of centralized control or configuration. CAN nodes only have to maintain a small amount of states that are independent of the numbers of nodes in the system, which make CAN scalable. When failures occur, a request can be routed to different path formed by alive intermediate peers.

The CAN design is a virtual d-dimensional Cartesian coordinate space. This space is partitioned among all peers in the systems such that each peer holds its individual zone within the space. Figure 1, which is deduced from [25], illustrates a example of 2-dimensional space. If we want to store a *(K, V)* pair, we use a uniform hash function to map key K onto a point P in the coordinate space. And suppose that peer E has zone containing the location of point P, the *(key, value)* pair is then stored at peer E. And peer E only needs to maintain the status of its 4 neighbors (A, B, C, D). At peer X, in order to retrieve an entry of corresponding key K, it uses the same hash function to map K onto point P. And because point P is not in peer X's zone and its neighbors, the request will be routed through the entire CAN structure until reaching the peer whose zone contain P. And the final destination is peer E.

CAN has an associated DNS domain name resolving to the IP address of one or more bootstrap nodes (A bootstrap node maintains spatial list of CAN peers). In order to join a CAN, peer Z looks up in the DNS to find a bootstrap peer's IP address. The bootstrap peer then randomly provides the IP addresses of chosen nodes currently in the system. After that,

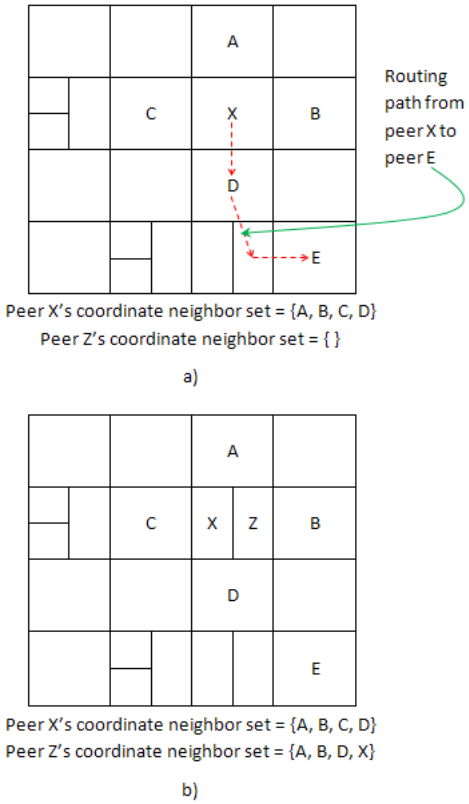

Figure 1: Example of 2d space CAN *a)* before and *b)* after Peer Z joins.

peer Z chooses a point P and send JOIN requests destined for point P. The destined zone (peer X) is splitted in half and the half is assigned to peer Z. The *(key, value)* pairs from the half zone to be handed over are also transfered to peer Z. After having its zone, peer Z learns the IP addresses of its neighbors from the previous occupant X. This set is the subset of peer X's neighbors, plus peer X itself.

CAN is a suitable candidate in large scale storage management systems which requires efficient insert and retrieval of content with a scalable indexing mechanism. Another potential application for CAN is in the construction of wide-area name resolution services which decouple the naming scheme from the name resolution process thereby enabling arbitrary, location-independent naming schemes.

2.2 Chord

The core operation of P2P protocol is the efficiency of lookup mechanisms. Each P2P protocol has its own lookup mechanism that affects its performance. Chord [18] protocol is one of the original DHT protocol which use consistent hashing to assign keys to it peers in a circle. In a Chord system, each peer receives the same number of keys and involves little movement of keys when nodes join and leave the system. Three features that distinguish Chord from other peer-to-peer retrieve protocol are simplicity, provable correctness and provable performance. A Chord node only need to maintain the information about O(logN) other nodes for efficient routing. But performance degrades when that infor-

mation is out of date. In practice, nodes will join and leave arbitrarily, thus consistency of O(log N) state may be hard to maintain.

In Chord, an identifier ring is used to stored nodes and keys. This ring is actually a circle of numbers from 0 to 2^{m-1}. On the ring, a node's identifier is calculated by hashing the node's IP address and a key identifier is produced by hashing the key. *m* is chosen to be large enough to make the probability of keys hashing to the same identifier negligible. Figure 2, which is deduced from [18], show an example of chord ring with *m*= 6. This ring has 10 nodes and stores 5 keys. Chord assigns keys to nodes as follows. Key *k* is assigned to the first node whose identifier is equal or follow *k* in the identifier circle. This is call the successor node of key *k*, denoted by *successor(k)*. The successor of key *k* is the first node clockwise from *k*. For example, the successor of identifier 10 is node 14, so key 10 is located at node 14. Similarly, keys 24 and 30 are located at node 32, key 38 at node 38, and key 54 at node 56. The consistent hashing provides minimum disruption when nodes enter or leave the network. When a node *n* joins the network, certain keys previously assigned to *n*'s successor are assigned to *n*. When node *n* leaves, all of its keys are assigned to *n*'s successor. The joining and leaving process requires only $O(\log^2 N)$ messages. No other changes of key assignments to nodes need to occur.

The query for a given identifier is passed around the circle through the successors' pointers. An example is shown in figure 2. Node 8 performs lookup for key 54. It performs the *find_successor* operation for this key. The query passes every nodes on the circle between node 8 and node 56. This eventually returns the successor of that key, node 56. However, this technique requires the amount of messages linear in the number of nodes. To accelerate the retrieve process, Chord using additional routing information. Each node *n* maintains the table with *m* entries. This table is called *finger table*. The ith entry in the finger table contains the identity of the first node *s* that succeeds *n* by at least 2^{n-1} on the identifier circle, i.e. $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$ (all mathematic is modulo 2^m). An entry in finger table contain Chord identifier and the IP address of the relevant node. Figure 2 show the finger table of node 8. The first entry is node 14, as node 14 is the first node that succeeds $(8+2^0) \bmod 2^6 = 9$. Similarly, the last entry is 42, as node 42 is the first node that succeeds $(8+2^5) \bmod 2^6 = 40$.

The successor pointers of peers change when a peer joins or leaves the systems. It is very important to keep the pointers updated as it will affect the correctness of retrieving keys. However, Chord has implemented a *stabilization* algorithm [18] that run periodically to update the pointers and entries in the finger table. When a node leaves, another nodes possibly don't know their successors and predecessors. To avoid this situation, a node maintain a list of *r* successors. When the successors node does not responds, the node contact the next node on the successor list. Suppose that the probability of failure on every node is *p* then the probability that all nodes in the list fails is p^r. By increasing the number of node *r* in the list, the system will be more robust.

Chord can be used in Cooperative File System (CFS) [9]. In this system, various providers of content cooperate to store and serve each others' data. The benefit of spreading

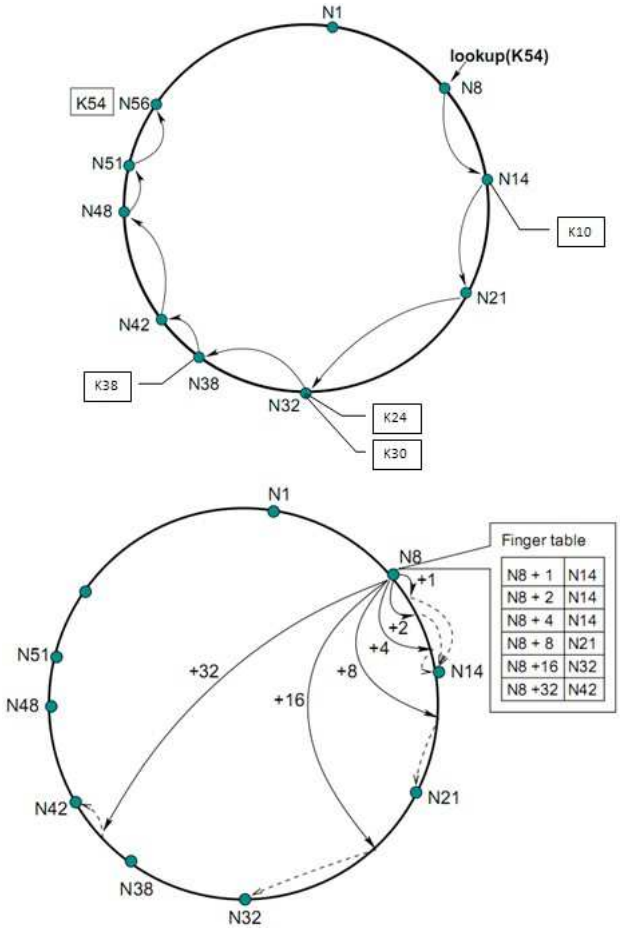

Figure 2: Chord ring with identifier circle consisting of 10 peers and five data keys. Finger table entries for peer 8.

total load evenly over all participants is the low cost of the system, because each participant only needs to provide capacity for the average load , not for the peak load. Another application is Chord-based DNS [10]. It provides a lookup service with hostname as keys and IP addresses as values. Chord provides DNS service by hashing each host name to a key. It requires no special server, compared to original DNS systems. Furthermore, it also automatically maintains the correctness of the analogous routing information.

2.3 Tapestry

The inspiration for Tapestry' s design is the location and routing mechanisms introduced by Plaxton, Rajamaran and Richa in [8]. Plaxton et al. present a distributed data structured, known as *Plaxton mesh*, optimized for locating name object and forwarding messages to those objects. It allows messages to locate objects and to be routed to them across an arbitrarily-sized network using a small constant-sized routing map at each node. In Plaxton mesh, nodes can take the roles of servers, which store data objects, and clients, which issue requests. It uses local routing map at each node, also known as *neighbor maps*, to route the message to the destination. This is done incrementally digit by digit of the destination ID (for example, $***0 \Longrightarrow **10 \Longrightarrow *410 \Longrightarrow 2410$, where '*' is the wildcard).

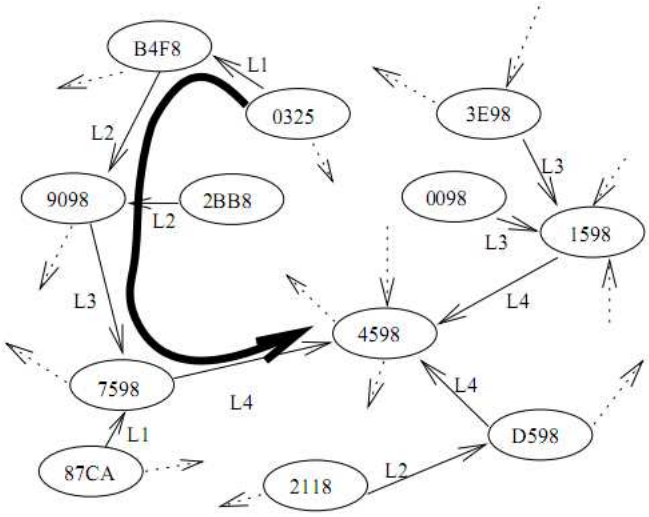

Figure 3: An example of routing in Plaxton mesh. Here we see the path taken by a message originating from node 0325 destined for node 4598 in a Plaxton mesh using hexadecimal digits of length 4.

An example is shown in figure 3 [14]. In our discussion, we resolve the digit from right to left, but the decision is arbitrary. A node *n* has a neighbor map with multiple levels. Each level represents a matching suffix up to a digit position in the node ID. A given level of the neighbor map contains a number of entries equal to the base of the ID, where the i^{th} entry in the j^{th} level is the ID and location of the closest node which ends in *"i" + suffix(n, j -1)*. For example, the 7^{th} entry of 4^{th} level for node 94A7C is the node closest to 94A7C which ends in 7A7C. The n^{th} node that the message reaches share a same suffix at least the length *n* of the destination ID. In order to find the next node, we look at its $(n+1)^{th}$ level map, and find the entry matching the value of the next digit of the destination ID. This routing method ensures that any node in the system will be found within at most $\log_B N$ hops, where N is number of nodes and B is the base of node ID. As every single neighbor map at a node assumes that the preceding digits all match the current nodeŠs suffix, only a small constant size B entries at each route level need to be kept. Thus a neighbor map of fixed constant size is: $(entries/map) \times number\_of\_maps = B \cdot (\log_B N)$

The goal of Tapestry is the ability to detect and recover from failures. The most common faults impacting routing are server outages (due to high load and hardware/software failure), link failure and neighbor map corruption. Tapestry can detect failures quickly, operate under them and recover routing state when the failures are repaired. To detect server and link failures, Tapestry utilizes TCP timeouts. Besides, the nodes send to their neighbors the UDP periodic heartbeat packets. This is simply a message that assures the reachability of the message source. By checking the ID of each arrived message, faulty and corrupted neighbor tables can be quickly detected. In order to operate under failures, every entry in the neighbor map maintains two backup neighbors beside the primary neighbor. When the primary neighbor fails, it will switch to the backup neighbors in order. Furthermore, Tapestry wants to avoid the costly reinsertions of a node when failures have been repaired. When a node detects that a neighbor is unreachable, it marks the neighbor as unreachable in stead of removing it pointer. It maintains a reasonable *second chance* period, during which the message is still routed to the failed node. If the failure can not be repaired in this period, the failed neighbor is removed from the map.

Tapestry provides an overlay routing network that is stable under a various network conditions. Thus it provides an ideal infrastructure for distributed applications and services. One application of Tapestry is the OceanStore system [19] [5], which is a global-scale, highly available storage utility deployed on the PlanetLab. Another applications include Bayeux [24], which is an efficient self organizing multicasting application, and SpamWatch [6] - a decentralized spam-filtering system using the similar search engine implemented on Tapestry.

2.4 Analysis and Comparision

In CAN network, peers are organized into a d-dimensional Cartesian coordinate space. Each peers have the ownership of a specific area on the space. A CAN node learns and maintains the routing table which holds the IP address and virtual address of neighbors. Suppose that we have a *d*-dimensional spaced divided into *n* zones, the average routing path length is $(d/4)(n^{\frac{1}{d}})$ hops and each node maintains *2d* neighbors. From this formula, we can understand that the number of nodes and zones can be increased without affects to the node states, and the path length grows as $O(n^{\frac{1}{d}})$. In order to join a CAN network, a peer need to connect to any existing peers. And then the region of that peer will be splitted into halves, the new peer take one half as its region. When a peer leave the network, it hand over its data such as NodeID, list of neighbors to a takeover peer. The routing performance in CAN network can be improved through various factors. First of all, it depends on the dimension of the space. Secondly, a peer can choose the neighbor closest to the destination in CAN space. Or the landmark-based placement can let peers to probe a set of well known landmark hosts, which estimate each of their network distances. There are still questions on CAN's resiliency, load balancing, locality and latency costs.

In Chord system, keys and peers are map onto an identifier ring. On the ring, a node's identifier is calculated by hashing the node's IP address and a key identifier is produced by hashing the key. *m* is chosen to be large enough to make the probability of keys hashing to the same identifier negligible. This *Consistent Hashing* help to minimize the disruption when a peer joins or leaves the network. It ensures that the number of caches for an object is limited and when these caches change, the minimum number of object references will move to maintain load balancing. Beside that, each Chord peer only needs to maintain information about O(logN) other nodes for efficient routing. In fact, Chord is similar to binary search, where the searching space is reduced half after a search/routing-hop. So in a N-node network, the number of nodes to contact to resolve a query is O(logN). As a extent of Chord, if we use K-ary tree search [26], the search hop will decrease to $O(log_k N)$,

while the items of routing table in each node will increase to $O((k-1) * log_k N)$. One significant process that also effects the operation of Chord is the background maintenance process. This *stabilization algorithm* runs periodically to update the pointers to successors and the entries in finger table. But how often the stabilization procedure needs to run to determine the success of ChordŠs lookups. Liben-Nowell et al. [1] already performed an analysis research to determine the optimum involves the measurements of peersŠ behavior based on this feature.

The location and routing mechanism of Tapestry are similar to those of Plaxton. Every node contains a neighbor map. It contains many routing levels, each of which contains entries pointing to closest nodes that matches the suffix for that level. Each node also maintains a *backpointer list* pointing to nodes where it refers as neighbors. We use them in the node integration algorithm to generate the appropriate neighbor maps for a node. Because the routing uses local data, Tapestry is inherently decentralized, by which we can avoid the bottleneck in the network. In addition, routing requires nodes match a certain suffix, it it possible to route around any failure by choosing another node with a similar suffix. However, Tapestry requires global knowledge at the time when the mesh is constructed. This complicates the process of adding and removing nodes from the network.

These DHT-based systems are susceptible from the malicious peers' attacks. These attacks originate from the peers that do not follow the protocol correctly, as describe in [16]. The malicious peers can eavesdrop the communications between peers. The IP address is the weak point of peer identity as the malicious peers can trick others to believe and send data objects to there IP addresses. The possible attacks involve routing deficiencies due to corrupted lookup routing and updates.; inconsistent behaviours of peers; denial service attack by overloading the victims' connections, and unsolicited responses to lookup queries. In [22], Castro et al. present a design and analysis of techniques for secure routing (also known as *Eclipse attack*), secure peer joining, routing table maintenance, and robust message forwarding in Structured P2P overlay networks. These techniques can tolerate up to 25% of malicious peers with a small compromised peer. But they can only work for Structured P2P overlay network. Thus, new technique is proposed in [11] to prevent Eclipse Attacks in both Structured and Unstructured network. When the attacker launches an Eclipse attack, the indegree of attacker nodes should be higher than the average indegree of correct nodes . Thus the correct nodes choose neighbors from nodes whose indegree and outdegree are below a threshold. An efficient auditing technique is also used to prevent attacker nodes from lying about their indegree and outdegree.

Figure 4 summarizes the features of Structured P2P overlay networks that have been discussed in section 2.

	CAN	Chord	Tapestry
Decentralization	DHT-based functionality		
Operation Architecture	Multi-dimensional space.	Uni-directional and Circular space.	Plaxton mesh network.
Retrieve Protocol	Using has function to map (key, value) onto coordinate space.	Matching key and nodeID.	Matching suffix in nodeID.
System Parameters	Number of peers N Number of dimensions d	Number of peers N	Number of peers N Base of peer identifier B
Routing Performance	$O(d.N^{\frac{1}{d}})$	$O(logN)$	$O(log_B N)$
Routing State	$2d$	$logN$	$log_B N$
Join/Leave Behavior	$2d$	$(logN)^2$	$log_B N$
Security	Suffers from man-in-middle and Trojan attacks.		
Reliability	Peers' failure does not cause network failure. Multiple peers are responsible for each data item.	Peers' failure does not cause network failure. Replicate data are on multiple consecutive peers.	Peers' failure does not cause network failure. Replicate data are on multiple peers.

Figure 4: A comparison of CAN, Chord and Tapestry network schemes.

3 Unstructured P2P Overlay Networks

In the unstructured P2P networks, there are no strict rules defining where data is stored or which nodes are neighbors of other nodes. They organize peers in a random graph in flat or hierarchical manners. To look up for specific data, peers use flooding, random walks or expanding ring Time-To-live (TTL). In this section, we will survey and compare two Unstructured P2P overlay networks: Freenet [17] and Gnutella [3] [27].

3.1 Freenet

Freenet is an adaptive peer-to-peer network of nodes that query one another to store and retrieve data, which are named by location-independent keys. Nodes in Freenet system maintain the local data files, which are available to read and write for the whole network, and a dynamic routing table that contains the IP addresses of other nodes as well as the key that they hold. The architecture of Freenet is totally decentralised and distributed, which means that there are no central servers and all computations and interactions happen between clients. Clients connect randomly to other available clients. This makes Freenet network an unorganised scattered topology. Requesting for keys are passed along nodes, in which nodes decide the next location to send. The request routing uses the style of IP routing. Each request has a hops-to-live limit, similar to the IP's time-to-live,which is decremented at each node to prevent infinite loops. Each request also has an identifier, thus nodes can prevent loops by rejecting requests they have seen before. When this situation occurs, a different node is chosen to forward the request. This process continues until the data is found or the request exceeds its hops-to-live limit. Then result is passed back to the sending node by the sending path.

Figure 5 shows an example of request sequence in Freenet. The user at node a initiates a request. Node a forwards the request to node b, which forwards it to node c. Node c is unable to contact any other nodes and returns a "request failed" message to b. Node b now forward request to node e, which forwards the request to f. Node f forwards the request to b, which detects the loop and returns a failure message. Node f is unable to contact any other nodes and backtracks one step further back to e. Node e forwards the request to its second choice, d, which has the data. The data is returned from d via e and b back to a, which sends it back to the user. The data is also cached on e, b, and a.

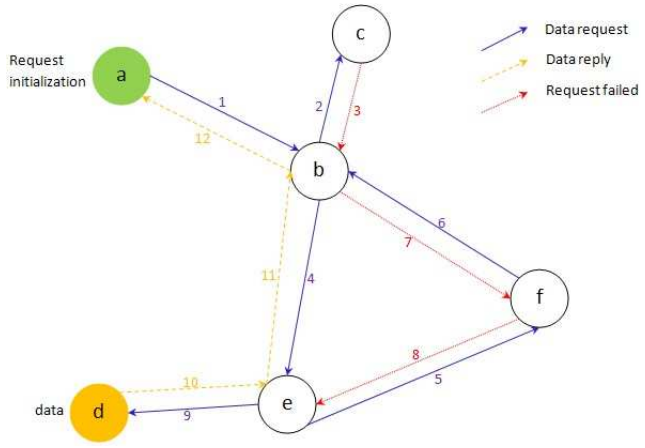

Figure 5: An example of a request sequence in Freenet.

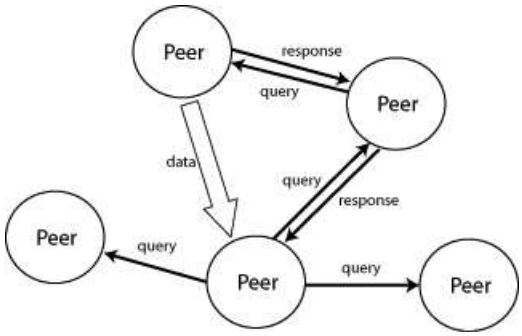

Figure 6: A typical query and respond in Gnutella.

The data files in Freenet are identified by binary file keys. We obtain the keys by using 160-bit SHA-1 [7] as our hash. Three different types of binary keys are used for various purposes. The simplest one is *Keyword-Signed Key* (KSK). This key is derived from the descriptive string the user choose to when store the files in the network. The string is fed to a function to get a public/private key pair. The public one is then hashed to yield the file key. The private one is used to sign the the file, which provide minimal check that a retrieved file matches its file key. But nothing prevents users to choose same descriptive text strings for different files. However, a second technique, called *Signed-Subspace Key* (SSK), solves this problem. A user creates a namespace by randomly generating a public/private key pair. The pair is used to identify his namespace. In order to insert a file, user chooses a short descriptive string. The namespace key and the descriptive string are hashed independently, then XOR'ed together, and finally hashed again to get the file key. This signature is generated from a random key pair which is more secure than in the *Keyword-Signed Key*. The file is also encrypted by the descriptive string. The third type of binary key is *Content-Hash Key* (CHK). This one is useful for implementing updating and splitting the contents. A hashing function is applied to the content of file to get the key, giving every file unique key. To allow others to retrieve the file, the user publishes the content-hash key together with the decryption key.

The Freenet provides an effective means of anonymous information storage and retrieval. It keeps information anonymous and available while remaining highly scalable by using many nodes with adaptive routing algorithm. Because of the anonymous feature, it is hard to show exactly how many users there are or how well the insert and request mechanisms are working. Thus Freenet also has the potential to create an anonymous network in which copyright and illegal information can be traded with no fear of reprisal.

3.2 Gnutella

Gnutella [27] is s decentralized peer-to-peer system consisting of hosts connected to each other over TCP/IP. In the Gnutella network, the traffic consists of: queries for data, replies to queries and discovering messages to find nodes. This type of network allows to share arbitrary resources (e.g. resources are mapping to other resources, meta-information and other types of pointers). If a node wishes to join the Gnutella network, it must connect to any existing nodes of the system. The mechanism *"host caches"* allow a new nodes join the network by connecting to a random nodes' IP address provided via DNS or on a specific website. The new node opens a TCP connection to the existing node and perform a handshake. If the connecting node refuses the connection, it will recommend other IP addresses for the new node to connect to. A node needs to connect to multiple existing nodes in order to reach the whole Gnutella nodes. When it manage to join the network, it communicates with neighbor nodes by sending and receiving Gnutella protocol messages and accepting connections from new nodes. The main protocol messages are:

- Ping: a request for information about another nodes.
- Pong: a reply carrying information about a node.
- Push: a mechanism that allows a firewalled node to share data.
- Query: a request for a resource.
- Query Hit: a response identifying an available resource.

Gnutella is a broadcast network, in which Pings and Queries are forwarded to multiple nodes. To reduce the resource consumption, nodes cache Pongs to response to Pings when they can. Pongs and Query Hits are routed back to the path needed to reach the destination. In this respect, the queries are inefficient due to flooding, but the replies is rather efficient. Figure 6 shows a typical query and respond in Gnutella network.

A node that wants to perform a search sends a Query message to al nodes connected directly to it. Each of them will then replicates and relays the Query message to its neighbors, except the node that originates the query. This process continues within some certain regions. And because of the flooding characteristic, there is a possibility that the network is in the bottleneck state. Thus the maximum size of Query message is limited to 256 bytes. A *Time-to-Live* (TTL) value is also assigned to the message to avoid the endless forwarding loop. A node replies with a Query Hit message when

it has content that satisfies the request. This Query Hit contains the IP address and port number where this specific node can be reached for the data transfer. When a node receives a Query Hit message, it knows where to get the data. In Gnutella, the data are downloaded out-of-network: instead of wasting the Gnutella network capacity, the two nodes involved in the transfer connect over TCP/IP and transfer the data directly. The file download protocol is HTTP/1.0 or HTTP/1.1. These are the standard protocols for downloading files from web servers. The node wanting to download a file makes a TCP/IP connection to the serving host at the IP address and port number specified, then makes a standard HTTP request.

Super-peer or ultrapeer [27] is a prominent feature of Gnutella system. With this idea, the peers are categorized into *"regular peers" (leafs)* and *"ultra-peer"*. An ultra-peer is a host with sufficient network bandwidth and can act as a proxy for a large number of connecting clients. The ultra-peer removes the burden of message routing from the client. It is used as an entry point into the network for the leaf nodes. The ultra-peer concept lets the Gnutella network scale quite well as it reduces the number of nodes actually involved in message routing. With this scheme, the Gnutella network mimics the Internet: low bandwidth nodes are connected to larger routers (the ultra-peers) that transmit the majority of the data over high bandwidth backbones.

3.3 Analysis and Comparision

The Unstructured P2P network is characterized by the flooding technique, which is quite efficient in finding popular data objects. However, it can not find rare item due to the limit of lookup TTL parameter and can results in excessive bandwidth consumption. Although DHT-based system have the ability to find are data and have efficient searching mechanisms, they are not widely deployed as their ability to handle unreliable peers has not been tested. Thus, there are till many efforts to improve the lookup properties of Unstructured P2P overlay system.

Gnutella and Freenet use simple protocols that let peers query one another in a chain. They make location irrelevant. The data in Gnutella and Freenet belongs to the whole system rather than to a particular node. Freenet aims at protecting anonymity, in which the distributing information cannot be traced and its location is irrelevant. The Freenet also introduces the indexing scheme where data objects are identified by *Content-Hashed Keys* or secured *Signed-Subspace Keys* to ensure that only object owners have writing access and other can only read it. While most people consider Gnutella and Freenet as the same, Gnutella is not designed for anonymity. When querying for data objects, Gnutella peers are likely to return a location (usually IP address and connecting port) or some other identifying information. Two Gnutella developers, Spencer Kimball and Gene Kan, explained that Gnutella is going beyond a normal file sharing by allowing the distributed processing of search queries, and thus better distributing information about whatŠs available online. In the Gnutella protocol version 6, the ultra-peers concept is introduced. These are high capacity peers that act as the proxies for other low capacity peers. The improvement is made based on the *Query Routing Protocol* (QRP), by which the leaf nodes can forward the query to the ultra-peers and reduce the lookup query traffic at the leaf nodes. As described in [29], peers are attached to high-degree peers that provide a receiver-based token flow control for sending lookup queries to neighbors. Instead of flooding, they use *random walk* search algorithm and the system also keep the pointers to objects in neighboring peers. Unstructured P2P overlay are proposed to be built on top of Structured P2P overlay to reduce the lookup queries overhead and overlay maintenance traffic.

	Freenet	Gnutella
Decentralization	Totally decentralized and distributed.	Topology is flat and distributed.
Operation Architecture	Data objects are identified by keywords and descriptive text strings.	Ad-Hoc network of peers.
Retrieve Protocol	Search from peer to peer using descriptive text string.	Flooding mechanism.
System Parameters	None	None
Routing Performance	Searching until exceeding the Hop-To-Live (HTL) limits.	Using Time-To-Live (TTL) to avoid endless loop forwarding.
Routing State	Constant	Constant
Join/Leave Behavior	Constant	Constant
Security	Low. Suffers from man-in-middle and Trojan attacks.	Low. Suffer from flooding malicious content, virus spreading and denial of service attacks.
Reliability	No central point of failure.	Performance degrades when number of peers grows.

Figure 7: A comparison of Freenet and Gnutella network schemes.

Most of these Unstructured P2P network are not pure power-law networks. For example, an analysis research in [21] shows that Gnutella networks have topologies that are power-law random graphs, and there are too few peers with a low number of connectivity. This might reflect the behaviors of the users of P2P networks. Many research on power-law networks shows that in diverse networks, most peers have few links while a few peers have a large number of links. Adamic et al. [20] study the random-walk search strategies in these power-law networks, and discover that by changing walkers to seek out high degree peers, the search performance can be optimized greatly. All the security issues discussed in the Structured P2P overlay section is applicable to Unstructured P2P overlay networks.

Figure 7 summarizes the features of Structured P2P overlay networks that have been discussed in section 3.

4 Hybrid Search in Peer-to-Peer Networks

Unstructured P2P network adopts flooding techniques to locate files. These techniques are effective for locate highly replicated data, but not for rare data. In contrast, Structured P2P networks are very good at locating rare item, but they get higher overloads for popular files. A hybrid searching [13] [12] [23] technique is proposed. By using this, structured search techniques are used to index and locate rare items, and flooding techniques are used for locating replicating items.

In Loo's experiment [13], he showed that queries for rare data in Gnutella have low recall rate. About 18% of

Gnuetella queries return no results despite there are results available in the system at two third of these queries. Such queries also have poor response time. On the contrary, DHT have good recall and response time for rare data while incur bandwidth cost for poppular item publishing. A hybrid P2P search combines unstructured flooding technique with structured DHT-based global index [13] [12]. Based on the popularity of items, queries are perfomed either by flooding the unstructured network, or by looking up in the DHTs. This will improve the recall and response time of queries for rare items with low bandwidth overhead, and furthermore maintain good recall and response time for highly replicated items.

The main issue in hybrid search is how to estimate the number of peers that can reply a query, such that we can determine the best search operation. We have two options for differentiating queries. The first one is detection-based, also known as *simple hybrid strategy*. In this type, a search is first performed by flooding. If there are not enough results returned within a predefined time, the query is resumed to a DHT query [12]. Although popular and rare data can be located, locating rare items has a worse response time than pure DHT. Besides extra cost is incurred by pre-flooding. The second type utilizes the information gathered through a gossiping method to estimate the popularity of the items [23]. Using a threshold to determine whether flooding or DHT, this approach outperforms simple hybrid techniques.

Hybrid search provides an efficient search for P2P systems. In order to further improve its performance, we have to better estimate how many peers can answer a query, so that a proper search strategy for the query can be determined.

5 Conclusion

This paper has presented various schemes in Structured and Unstructured P2P overlay networks. Furthermore, we compare the characteristics of each scheme to see how they can actually interact with each other. Peer-to-peer applications attracted millions of users in a short period of time. Today, around 60% to 80% of the Internet traffic originates from applications within the peer-to-peer paradigm. The most popular applications are file sharing and content distribution. Peer-to-peer will still remain attractive for researchers as central entities can be avoided, straight forward communications between the nodes, and peer-to-peer requires autonomy of nodes. In order to move forward, Peer-to-peer demands new ideas from the researchers, new ways of thinking and the significant combination of emerging mechanisms and existing technologies.

References

[1] *Analysis of the evolution of peer-to-peer systems*. [Internet] `http://nms.lcs.mit.edu/papers/podc2002.pdf`. [Accessed 6 Apr 2009].

[2] Bittorrent. [Internet] `http://www.bittorrent.com/`. [Accessed 9 Feb 2009].

[3] Gnutella development forum. [Internet] `http://groups.yahoo.com/group/the_gdf/`. [Accessed 2 March 2009].

[4] Napster. [Internet] `http://free.napster.com/`. [Accessed 9 Feb 2009].

[5] Ocean store. [Internet] `http://oceanstore.cs.berkeley.edu/`. [Accessed 25 Feb 2009].

[6] Spamwatch. [Internet] `http://www.cs.berkeley.edu/~zf/spamwatch/`. [Accessed 25 Feb 2009].

[7] SECURE HASH STANDARD. Technical report, Federal Information, Processing Standards Publication 180-1, April 1995. [Internet] `http://www.itl.nist.gov/fipspubs/fip180-1.htm`. [Accessed 18 Feb 2009].

[8] *Accessing nearby copies of replicated objects in a distributed environment*, 1997. In Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures [Internet] `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1850`. [Accessed 24 Feb 2009].

[9] *Widearea cooperative storage with cfs*, October 2001. In Proceedings of the eighteenth ACM symposium on Operating systems principles[Internet] `http://pdos.csail.mit.edu/papers/cfs:sosp01/`. [Accessed 18 Feb 2009].

[10] *Serving dns using chord*, March 2002. In Proceedings of the First International Workshop on Peer-to-Peer Systems [Internet] `http://pdos.csail.mit.edu/papers/chord:dns02/index.html`. [Accessed 18 Feb 2009].

[11] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. Technical report, September 2004. [Internet] `http://research.microsoft.com/en-us/um/people/mcastro/publications/eclipse.pdf`. [Accessed 9 March 2009].

[12] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing P2P File-Sharing with an Internet-Scale Query Processor. Technical report. [Internet] `http://www.vldb.org/conf/2004/RS11P2.PDF`. [Accessed 6 March 2009].

[13] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellersteins. The Case for a Hybrid P2P Search Infrastructure. Technical report. [Internet] `http://db.cs.berkeley.edu/papers/iptps04-hybridsearch.pdf`. [Accessed 6 March 2009].

[14] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. Technical report, IEEE Journal, January 2004. [Internet] `http://citeseerx.ist.psu.edu/`

viewdoc/summary?doi=10.1.1.13.8778. [Accessed 13 Feb 2009].

[15] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, April 2001. [Internet] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.1577. [Accessed 13 Feb 2009].

[16] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. Technical report. [Internet] http:\\pdos.csail.mit.edu/chord/papers/sec.pdf. [Accessed 9 March 2009].

[17] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. Technical report, July 2000. [Internet] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.4919. [Accessed 28 Feb 2009].

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. Technical report, ACM SIGCOMM, 2001. [Internet] http://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf. [Accessed 13 Feb 2009].

[19] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. Technical report, November 2000. [Internet] http://oceanstore.cs.berkeley.edu/publications/papers/pdf/asplos00.pdf. [Accessed 25 Feb 2009].

[20] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman,. Search in power-law networks. Technical report, 2001. [Internet] http://www.hpl.hp.com/research/idl/papers/plsearch/pre46135.pdf. [Accessed 6 Apr 2009].

[21] M. A. Jovanovic, F. Annexstein, and K. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical report. [Internet] http://citeseer.ist.psu.edu/old/747051.html. [Accessed 6 Apr 2009].

[22] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. Technical report, December 2002. [Internet] http://www.cs.rice.edu/~dwallach/pub/osdi2002.pdf. [Accessed 9 March 2009].

[23] M. Zaharia and S. Keshav. Gossip-based Search Selection in Hybrid Peer-to-Peer Networks. Technical report. [Internet] http://iptps06.cs.ucsb.edu/papers/Zaharia-gossip06.pdf. [Accessed 6 March 2009].

[24] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. Technical report, June 2001. [Internet] http://oceanstore.cs.berkeley.edu/publications/papers/pdf/bayeux.pdf. [Accessed 25 Feb 2009].

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. Technical report, ACM SIGCOMM, 2001. [Internet] http://conferences.sigcomm.org/sigcomm/2001/p13.html. [Accessed 13 Feb 2009].

[26] S.E.Ansary, L.O.Alima, P.Brand, and S.Haridi. A framework for peer-to-peer lookup services based on k-ary search. Technical report, April 2003. [Internet] http://dks.sics.se/pub/TR-kary.pdf. [Accessed 6 Apr 2009].

[27] T. Klingberg, R. Manfredi. Gnutella 0.6. Technical report, June 2002. [Internet] http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html. [Accessed 2 March 2009].

[28] T. Tanner. Distributed Hash Tables in P2P Systems - A literary survey. Technical report, Telecommunications Software and Multimedia Laboratory - Helsinki University of Technology, April 2005. [Internet] http://www.tml.tkk.fi/Publications/C/18/tanner.pdf. [Accessed 11 Feb 2009].

[29] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. Technical report, August 2003. [Internet] http://berkeley.intel-research.net/sylvia/1103-chawathe.pdf. [Accessed 12 March 2009].