XYZ: A Scalable, Partially Centralized Lookup Service for Large-Scale Peer-to-Peer Systems

Jianying Zhang and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
Email: jzhang2@fau.edu, jie@cse.fau.edu

Abstract

Peer-to-Peer (P2P) systems are characterized by direct access between peer computers, rather than through a centralized server. File sharing is the dominant P2P application on the Internet, allowing users to easily contribute, search and obtain content. P2P systems can be categorized by the degrees of centralization. For fully centralized systems, the lookup service will not be available when the central directory server is down. For purely decentralized and unstructured systems, since there is no information about which nodes are likely to have the relevant files, searching essentially amounts to random search. This makes the lookup service unscalable and unpredictable. The objective of this paper is to design a partially centralized, scalable and self-organizing lookup service (XYZ) for wide area P2P systems. A clustering method is used to create the system backbone by connecting the cluster heads together and a color clustering method is adopted to create color overlays and minimize the searching space. Simulations and analysis are also provided. Extensions are proposed to achieve better performance.

1. Introduction

File sharing is the dominant P2P application on the Internet, allowing users to easily contribute, search, and obtain content. It raises many interesting research problems in distributed systems. This paper is focused on one of them, the lookup problem: How do you find any given data item in a large P2P system in a scalable manner, without any centralized servers or hierarchy? Traditionally, there have been two flavors of P2P lookup systems.

One approach is to maintain a central database that maps a file name to the locations of servers that store the file. Napster (http://www.napster.com/) adopts this approach for song titles, but it has inherent reliability and scalability problems that make it vulnerable to attacks on the database. Another approach, at the other end of the spectrum, is for the consumer to broadcast a message to all its neighbors with a request for a file. Gnutella has a protocol in this style with some mechanisms to avoid request loops. However, this "broadcast" approach doesn't scale either [16], because of the bandwidth consumed by broadcast messages and the computing cycles consumed by the many nodes that must handle these messages.

To reduce the cost of broadcast messages, one can organize the nodes in the network into a hierarchy, like the Internet's Domain Name System (DNS) does. The disadvantage of the hierarchical approach is that the nodes higher in the tree take a larger fraction of the load than the leaf nodes, and therefore require more expensive hardware and more careful management. The failure or removal of the tree root or a node sufficiently high in the hierarchy can be catastrophic.

The lookup service will not be available when the central directory server is down for fully centralized systems. For purely decentralized and unstructured systems, since there is no information about which nodes are likely to have the relevant files, searching essentially amounts to a random search. This makes the lookup service unscalable and unpredictable.

In this paper we propose a partially centralized, scalable and self-organizing lookup service (XYZ) for wide area P2P systems. We use a clustering method to create the system backbone by connecting the clusterheads together and use a color-clustering method to create color overlays and minimize the searching space. Since XYZ is not fully centralized, you cannot shut it down by simply disabling the central directory server. Since XYZ is not purely decentralized, searching is guaranteed to be complete in a certain number of steps. Node joins and node departures are also adaptive because XYZ is self-organizing.

The contributions of this paper are as follows: We briefly introduce some traditional P2P lookup algorithms. We propose a new hybrid approach called XYZ which is partially centralized, scalable and self-organizing. We conduct extensive simulation and compare performance with some existing systems.

The remainder of this paper is organized as follows: Section 2 discusses several recent P2P lookup algorithms that have

provable guarantees including DHT and YAPPERS. Section 3 proposes the design of XYZ. Performance analysis and simulation are presented in section 4, and the paper concludes in Section 5.

2. Related Work

2.1. Types of P2P lookup

Traditionally, there are two flavors of P2P lookup algorithms. The first kind consists of Gnutella-style networks. These networks do not organize the content in the networks. Consequently, answering a total lookup requires flooding the entire network to search every node. SBO [21] is one of them. In SBO, all peers are separated into two groups, red and white. Each red peer builds a minimum spanning tree (MST). Unlike pure flooding, queries will only be forwarded along the MST. Although the total traffic and response time of the queries can be reduced by SBO, the queries are flooded to all the peers and it does not reduce the searching space. The nature of flooding makes the lookup service inefficient and unpredictable.

In contrast to Gnutella-style networks, several research groups have recently developed algorithms for the lookup problem that present a simple and general interface, a distributed hash table (DHT). Some examples of DHT systems are CAN [6], Chord [7], and Pastry [8]. The DHT interface is built on top of an arbitrary overlay network to provide efficient querying. A unique key is assigned to each data item and stored in a DHT. To implement a DHT, the underlying algorithm must be able to determine which node is responsible for storing the data associated with any given key. To solve this problem, each node maintains information (e.g., the IP address) of a small number of other nodes ("neighbors") in the system, forming an overlay network and routing messages in the overlay to store and retrieve keys. In the rest of this section, we review some existing DHT-based lookup algorithms.

2.2. YAPPERS

Given the advantages and disadvantages of the Gnutella-style networks and DHT-based systems, Ganesan, Sun, and Garcia-Molina developed a hybrid system, YAPPERS [12], that operates on top of an arbitrary overlay network, just as Gnutella does, while providing DHT-like search efficiency.

Intuitively, YAPPERS works as follows: The key space of all the keys that need to be stored is partitioned into a small number of buckets. Each bucket has a unique color. Each node in the network is also assigned a color. Each node can only store keys that have the same color, so a query for a white key needs to be forwarded only to white nodes in the network.

YAPPERS divides a large overlay network into many small and overlapping neighborhoods (the immediate neighborhoods). The immediate neighborhood of a node A, denoted by $IN(A)$, is the set of nodes where A may store its <key, value> pairs. The data within each neighborhood is partitioned among the neighbors like a distributed hash table. When a lookup occurs and the neighborhood cannot satisfy the request, YAPPERS intelligently forwards the request to nearby neighborhoods, or the entire network if necessary. These forwarding require each node to know a larger set of nodes (the extended neighborhood $EN(A)$) that covers its neighbors' neighbors.

More generally, if the radius of the immediate neighborhood of a node is h, then it is necessary for the node to know all its neighbors within $(2h+1)$ hops of the extended neighborhood in order to guarantee that we can jump though all the nodes with the same color without touching any node with a different color.

3. XYZ

3.1. Basic ideas

A network can be logically represented as a set of clusters. The key space of all the keys that need to be stored is partitioned into a small number of buckets. Each bucket has a color. Every node is also assigned a color according to its node ID. The lookup strategy is to connect every node with the same color in the entire network through virtual circuits to form overlays and broadcast lookups within the overlay that have the same color as the key. So if the key space is partitioned into m buckets, then there are m overlays. A lookup for a red key will be broadcast within the overlay formed by red nodes. The path taken by the virtual circuit can change without affecting the overlay.

3.2. System design

❖ Data Structures

Every node belongs to a cluster of a CH (Cluster Head), which is at most k hops away. Every node is also a part of a color cluster of a CCH (Color Cluster Head), which is at most k hops away. So, each node has a member table in which it stores its CH ID and CCH ID (if the node is not a head), or cluster members' IDs and neighboring CHs and CCHs' IDs (if the node is a head or boundary node).

Table 1 (a) shows a member-table stored at some non-head node. Table 1 (b) is member-table stored at some CH. Table 1 (c) is member-table stored at some CCH. Table 1 (d) is member-table stored at some boundary node.

❖ XYZ Construction

An XYZ system can be constructed in 2 stages.

▪ *Stage 1: Clustering the whole network*

Table 1
Member tables

Member Type	Member ID
CH	1
CCH	13
CCH	2
:	:
CCH	7

(a)

Member Type	Member ID	Color	CCH
cluster	12	Black	Yes
cluster	23	White	No
:	:	:	:
NeighboringCH	4	Black	Yes
NeighboringCH	9	White	Yes

(b)

Member Type	Member ID
Color cluster	42
Color cluster	59
:	:
NeighboringCCH	11

(c)

Member Type	Member ID
CH	17
CCH	21
NeighboringCH	30
NeighboringCH	7

(d)

In the first stage, nodes are partitioned into k-hop clusters using the Lowest-ID algorithm [1], [2], and [3]. Each node in the network broadcasts its node ID and clustering decision exactly once. Each time a node receives a broadcast message initiated by a node within its $(k-1)$-hop neighborhood, it forwards it to all its neighbors.

Theorem 1: After k rounds, each node gets all the node IDs within its k-hop neighborhood.

Theorem 1 is obvious if each node is willing to forward. It can also be proved by induction. After round one, each node knows its direct neighbors' node IDs. After round two, each node receives node IDs of its two-hop neighbors forwarded by its direct neighbors. In this way, every node ID will be propagated in its k-hop range.

In the next step, all nodes whose ID is the lowest among all their k-hop neighbors broadcast their decision to create clusters. Node may hear the broadcasts by its neighbors. Then each selects the lowest ID among neighboring CHs, if any, and broadcasts the decision. If all neighbors who have a lower ID send their decisions and none declare itself a CH, the node decides to create its own cluster and declare itself a CH. Thus each node broadcasts its clustering decision after all its k-hop neighbors with lower IDs have already done so. Every node belongs to only one cluster. Clusterheads elected in this step are also called global CHs. Figure 1 shows 1-hop clusters produced by the lowest-ID algorithm.

In Figure 1, nodes 1, 18, 5, and 14 have the lowest IDs in their 1-hop neighborhood, so they declare themselves as CHs. Then their 1-hop neighbors join the neighboring CH with the lowest ID. Once node 2 decides to join node 1's cluster, node 3 can claim itself as CH. And once node 6 decides to join node 5's cluster, node 10 can claim itself as CH.

Theorem 2: Each node will eventually make a decision and joins only one cluster.
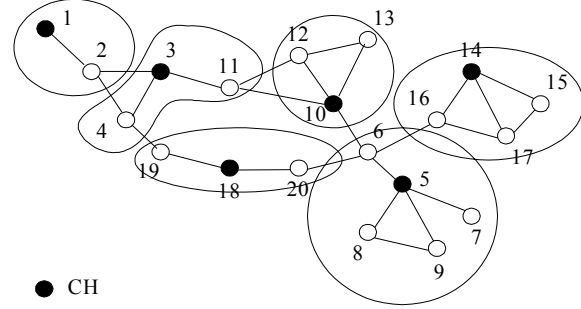
Figure 1. Lowest-ID clustering

Proof: According to the algorithm, nodes with the lowest IDs within their k-hop neighborhood are eligible to declare themselves CHs. Then nodes with a higher ID can make decisions. Since there are finite nodes in each node's k-hop neighborhood, there are finite nodes with a lower ID. So, a node is eligible to make a decision after finite steps. Although there may be more than one CH in a node's k-hop neighborhood, there is only one CH with the lowest ID. So a node will only join one cluster.

After the first two steps, each node knows the clustering decisions made by all neighbors. A node is a boundary node if one of its immediate neighbors belongs to a different cluster. All the boundary nodes send their neighboring clusterhead ID to their clusterhead. The network backbone can be easily set up by connecting all the neighboring clusterheads together. In Figure 2, node 24 tells its clusterhead node 3 that node 5 and node 6 are neighboring clusterheads. Node 80 tells node 3 that node 6 and node 7 are neighboring clusterheads. A backbone can be set up and works properly as long as the network is a connected graph.

- *Stage 2: Clustering nodes with the same color*

In the second stage, nodes within k hops with the same color are partitioned into color clusters. This stage is the same as stage 1, but only nodes with the same color and directly connected will be involved. Here the directly connected means no intermediate nodes with other colors. For example, in Figure 2, white node 54 has to connect white node 92 through some black nodes, so they can not do color clustering together. Instead, node 92 will do color clustering with node 42 and 50, and node 54 will do color clustering with node 68. Also, nodes do not need to broadcast node IDs again since we can use the neighbor info gathered from stage 1. Clusterheads elected in stage 2 are denoted as CCHs.

Theorem 3: Each node will eventually make a decision and join only one color cluster.

Proof: The proof of Theorem 3 is similar to that of Theorem 2, discussed above.
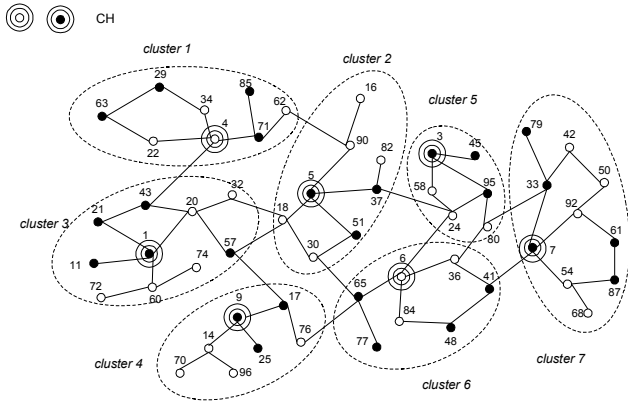
Figure 2. 2-Hop clustering

Figure 3. Network backbone

Next, CCHs with the same color are connected through virtual links to form an overlay. Every CH knows all the CCHs in its cluster and all the neighboring CHs. Every CCH sends a request to its CH. The CH will respond with all the other CCHs' IDs with the same color and forward the request to all the neighboring CHs if it has not forwarded such a request from a CCH with the same color yet. Once a CH receives a forwarded request from a neighboring CH, it will send the corresponding CCHs in its cluster to the generator of the request and stop forwarding the request.
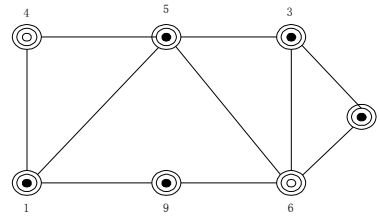
By doing this, CCHs within the same cluster can be connected easily. Then neighboring CCHs with the same color can also be connected. But all the CCHs with the same color may not be connected as one component if some global cluster does not have any CCH of that kind of color. For example, if cluster 6 in Figure 4 does not exist, then white CCHs 54 and 42 are connected as a component. The other white CCHs 4, 60, 16, 18, 82, 14 and 76 are connected as another component. Obviously these white overlay components are disconnected since there is no white CCH in global cluster 5.

In order to solve this problem, the CHs that do not have any of the requested CCHs must work as relay nodes and forward the request to neighboring CHs. The request will jump between global CHs until a CH which has CCHs of the requested color is found. Since all the clusters are connected, any two disconnected CCHs with the same color can get connected through intermediate global CHs. So, all the CCHs with the same color can get connected to form a color overlay.

Theorem 4: A color overlay can be formed for each color by connecting all the CCHs with the same color together.

Because all the global CHs are connected as a backbone, all the CCHs with the same color can find each other by checking CHs on the backbone.

❖ System maintenance

Node join: When a node joins the network, it asks its immediate neighbors about their CHs to see if there is any global CH within *k*-hop range. It joins the cluster of the CH within *k*-hops that has the lowest ID if more than one such CH exists. If there is no CH within its *k*-hop range, it will claim itself as a CH to form a new cluster, which consists only of itself. One of the new node's immediate neighbors that has the same color will also send its CCH ID to the new node. If the CCH is within the new node's *k*-hop range, it will select the CCH as its CCH; otherwise it will claim itself as CCH. If the new node has no neighbor with the same color, it will ask its CH for a CCH. The CH will either send a CCH in its cluster if it has such one or ask a neighboring CH to send one to the new node if itself does not have such CCH. In this case, the new node claims itself as CCH. Once it has selected its CH and CCH, it broadcasts its decision within the *k*-hop neighborhood. It also sends a join message to neighboring CHs to join the network backbone if it is a CH, and sends a join message to neighboring CCHs to join the corresponding color overlay if it is a CCH.

Node departure: If the node that left was a global CH, then all the nodes within its cluster need to do clustering again. Once all the new clusterheads are selected, they send a message to neighboring CHs to repair the backbone. If the node that left was a CCH, then all the nodes in its color cluster need to do color clustering again. Any new color clusterhead that is selected sends a message to its global CH to seek neighboring CCHs and sends a join message to them to join the corresponding color overlay. Node failure can also be considered as node departure. In case a node failure breaks the connected network into 2 parts, all the nodes in the broken joined cluster will either join neighboring clusters or select a new CH and CCHs. The new CH and CCHs will connect to neighboring CHs and CCHs to be part of the backbone and color overlay. You can see that XYZ is self organizing. Separated networks will work as separated XYZ systems. Separated backbones and overlays will remain functional.

We can reconstruct the XYZ system periodically, i.e., recluster the network, to maintain system consistency. This is because run-time maintenance, the schemes of node joins and node departures, is designed to be simple and affect nodes as little as possible to maintain the network efficiency. It should be noticed that maintenance overhead only applies to the small
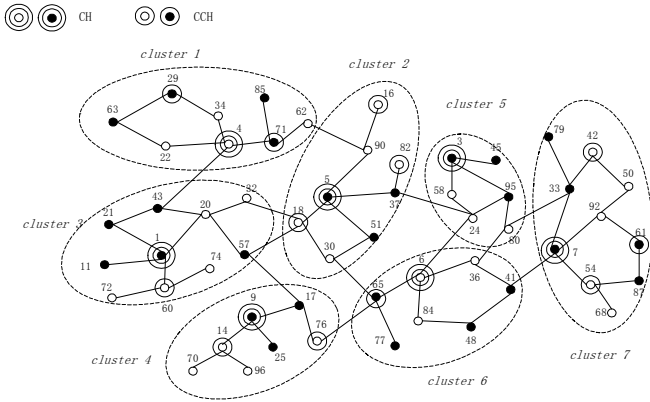
Figure 4. 2-Hop Clustering with CCHs labeled

Figure 5. System overview

range of affected nodes and the rest of the system is still functional. But by doing this, the resulting clusters may not be consistent with the clusters clustered using lowest-ID algorithm on all the nodes in the network. Although it is not necessary, in order to keep the consistency, we can always cluster the whole network using the lowest-ID algorithm periodically.

3.3. Case study

In this case, we use 2 colors and do 2-hop ($k = 2$) clustering in the whole network and color overlays. A node is a white node if its ID is even. A node is black if its ID is odd. Figure 2 shows 2-hop clusters produced by the lowest-ID algorithm. Figure 4 shows 2-hop clusters with CCHs labeled. Every boundary node informs its CH of all its neighboring CHs. For example, in cluster 1, node 62 is a boundary node. It informs its CH node 4 that node 5 is a neighboring CH. CH node 4 is also a boundary node. So it knows node 1 is another neighboring CH. Once every CH has figured out which other nodes are its neighboring CHs, connecting those CHs together can easily set up the network backbone. Figure 3 is the backbone of the network in Figure 2.

After all the color clusters are constructed, all the CCHs with the same color are connected to form overlays. At first, all the CCHs with the same color in a cluster are introduced by their CH, so the connections between those CCHs can be set up easily. For example, CCH 29 and 71 are 3 hops away. They do not know each other. But they are both in cluster 1. It is CH 4's responsibility to introduce them. Then, the CCHs with the same color in different clusters are connected by exchanging CCH information between neighboring CHs.

Figure 5 shows the topology of the XYZ system formed by color overlays. When node 94 joins the network, it checks its immediate neighbors, node 25 and node 77, for their CHs. Node 25 returns 9 and node 7 returns 6. Since only CH node 9 is within its 2-hop range, it will join cluster 4. Since it has no white
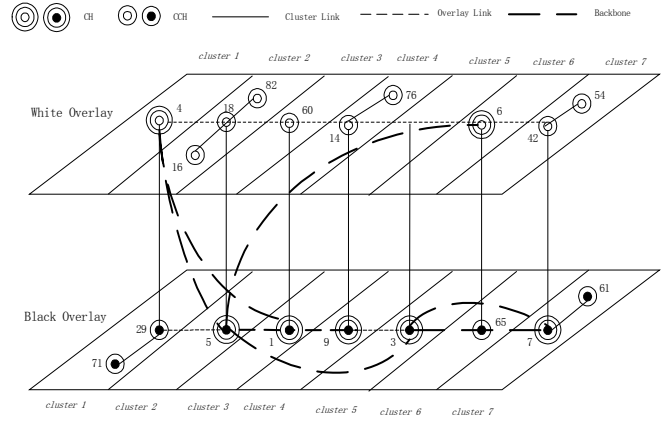
neighbors, it will claim itself as a CCH and asks its CH node 9 for neighboring white CCH. Node 9 will return either 14 or 76. Then the new node 94 will send its decision to node 9 and node 14 or node 76 to join cluster 4 and the white color cluster. If node 7 leaves the network, all the nodes in its cluster will do clustering again because it is a CH. Nodes 33, 92 and 54 are the new selected CHs. Three new clusters are formed in this case. An interesting thing is all the new CHs are CCHs except node 92. That is because its 2-hop white neighbor node 42 has a smaller ID.

3.4. Lookups

Once the system is constructed, it is easy to do lookup. If a node is searching for a file that has the same color, it will send the lookup (file_key) message with a timestamp and its ID to its CCH. The CCH will forward the message to neighboring CCHs in the same overlay. Any CCH that receives the lookup message will broadcast it in its color cluster and forward it to neighboring CCHs in the same overlay, except the one that forwarded the message to it. Any node that has the file will reply to the originator of the lookup message. Then, the lookup generator can selectively download the file from one or many hosts.

If a node is searching for a file which has a different color, it simply sends the lookup (file_key) message with a timestamp and its ID to its CH. The CH forwards the message to a corresponding CCH in the same cluster, if any, or forwards it to a neighboring CH and asks the CH to forward the query to a corresponding CCH in its cluster. Once a CCH receives the lookup message, it floods it within the overlay it belongs to.

Theorem 5: The node that has the requested file will be found if it exists.

All the nodes with the same color are connected by virtual links in the color overlay. Once the lookup message reaches one
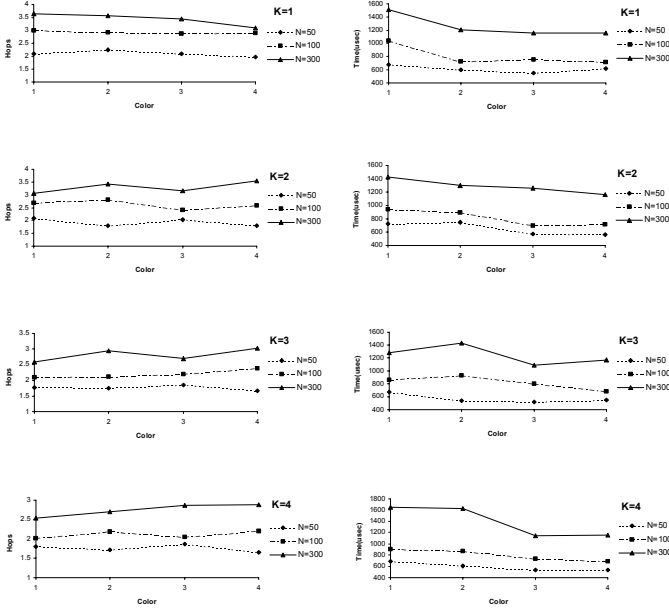
Figure 6. Fixed K

Figure 7. Fixed C

node in the overlay, it can also reach others in the same overlay. A timestamp and the originator ID can be attached with the request, so duplicate requests will not be forwarded at any nodes.

4. Performance Analysis and Simulation

4.1. Performance analysis

As mentioned in section 3, the system construction process consists of 6 steps. In the first step, each node propagates IDs of nodes within k-1 hops. Then every node will receive the IDs of all the nodes in its k-hop neighborhood. So the time complexity of this step is O(k). The message complexity is dependent on the network density. If every node has C k-hop neighbors in average, then each node will send out C messages in each round. So the message complexity is O(Ckn).

In the second step, all nodes whose ID is the lowest among all their k-hop neighbors declare themselves CHs; all nodes whose ID is the lowest among all their k-hop neighbors with the same color (no intermediate nodes with other colors) declare themselves CCH; Each node broadcasts its clustering decision and color clustering decision after all its k-hop neighbors with lower IDs have already done so. In the best case, every CH has the minimum ID within its k-hop neighborhood, so the time complexity is O(k). In the worst case, CHs claim themselves as clusterheads one by one. These CHs do not claim themselves as clusterheads until all the k-hop neighbors with lower IDs have made decisions. So in this case, the time complexity is O(d), where d is the network diameter. The message complexity is the
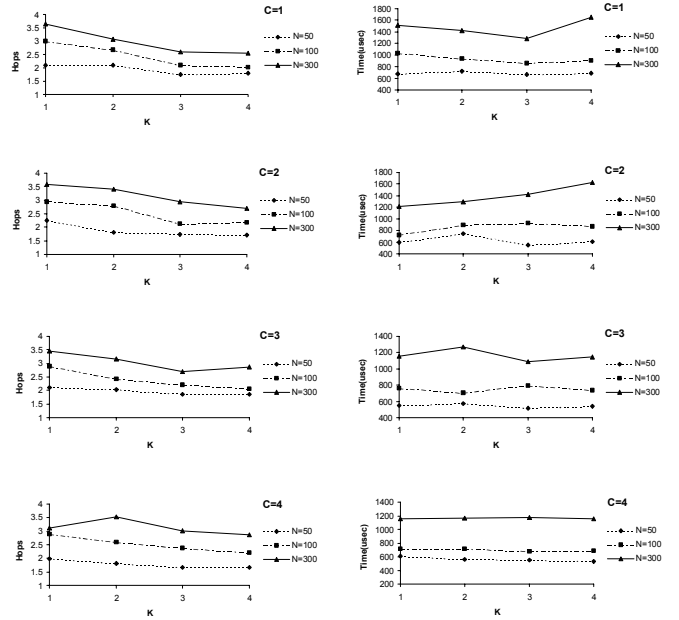
same as it is in step 1.

In the third step, all the boundary nodes send their neighboring clusterhead ID to their clusterhead. The network backbone is setup by connecting neighboring CHs together. Since boundary nodes are at most k hops away from their CHs, the time complexity is O(k). If the average node degree is r, then each CH has at most rk boundary nodes. So the message complexity is O(hrk) where h is the number of CHs.

In the fourth step, each CH introduces CCHs with the same color in the same cluster to each other, so that CCHs with the same color in a cluster are connected. Because the distance between the CH and any CCH in the same cluster is at most k, the time complexity is O(k). The message complexity is dependant on the number of CCHs, which is at most n, i.e., O(n).

In the fifth step, neighboring CHs exchange CCH information so that CCHs in different clusters can be connected to form a color overlay. Since 2 CHs are at most $2k$ +1 hops away, the time complexity is O(k), and the message complexity is O(h), where h is the number of clusters.

In the last step, disconnected CCHs with the same color are connected to form a color overlay. If the network diameter is d, then the distance between any 2 CCHS is at most d. So, the time complexity is O(d), and the message complexity is O(n), because there are at most n intermediate nodes.

In the XYZ system, the value of k, the radius of a cluster, and the total number of colors will also affect the lookup performance. If you increase the value of k, then the total number of clusters will be decreased and the distance between clusterheads will be increased, so messages can jump quickly within the network. But each CH will store more information
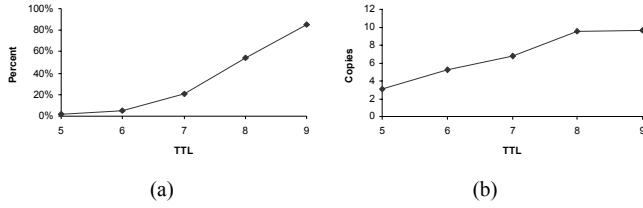
**Figure 8. (a) Percentage of connected color overlay
(b) Number of copies found for different TTL**

about its members since there are more nodes in its cluster. If we increase the total number of colors, then the number of color overlays will be increased and the number of nodes in each color overlay will be decreased. So the lookup can be done in less time since fewer nodes will be visited. But the disadvantage is more overhead will be introduced to maintain more overlays.

4.2. Simulation results

Simulations have been done on a single station. In the simulations, the range of k was selected from 1 to 4 and the range of color was selected from 1 to 4. There are three kinds of graphs being used in the simulations: 50-node uniform graph, 100-node uniform graph, 300-node uniform graph.

For each pair of k and color, 800 random lookups were performed in each graph, then the average lookup time and average jumps between the lookup initiator and the file hosts were computed. Figure 6 fixes k and compares the lookup time and jumps for different numbers of colors. Figure 7 fixes the number of colors and compares the lookup time and jumps for different k.

The following metrics are used to evaluate the system performance:
1) *Hops*: the average number of hops/jumps required to route the lookup message between the lookup initiator and the file host.
2) *Time*: the average latency between when the lookup initiator sends out the request and receives the response from the file host.

The key design parameters affecting system performance are:
1) k: number of hops used to do clustering. This is also the radius of each cluster.
2) *color*: the total number of colors used to group all the nodes
3) N: number of nodes in the system.
4) *density*: average number of neighbors of each node.
5) d: network diameter.

In a given network, N, *density* and d are all fixed. So the lookup performance depends on k and *colors* only.

In Figures 6 and 7, it is shown that no matter what combination of k and color are used in the simulation, the average hops and time of lookups in large networks are greater

than those values in small networks. That is because the network diameter increases, and more nodes are involved in the lookup process. In Figure 6, it is shown that when k is fixed and color increases, the average lookup time gets smaller. That is because fewer nodes will be touched. The average number of lookup hops does not change too much, since if k is fixed, the number of hops between the lookup initiator and the host depends only on the distance between them. Suppose both the source and destination are CHs and the distance between them is $3k$, then the number of hops is $3k/k$, which is 3. In Figure 7, it is shown that when color is fixed and k increases, the average number of lookup hops decreases. That's because the radius of each cluster is larger, which means lookup messages can jump further between clusters. But in this case, the lookup time does not change too much because with a fixed color, the number of targeted nodes is fixed. All the lookup messages will be flooded in the same group of nodes for different values of k.

Simulation results can be summarized as follows: When k is fixed and color increases, the average look-up time gets smaller but lookup hops does not change too much. When color is fixed and k increases, the average lookup hops gets decreased but the lookup time does not change too much.

We also did some simulations on a modified XYZ system. Compared with the original XYZ system construction process, we do not do the global clustering in the modified XYZ system. What we do is color cluster the whole network. Then each CCH broadcasts its role using a TTL (time to live). By doing this, all the neighboring CCHs with the same color are known by each other and hence they can be connected as a color overlay. The problem with the modified XYZ is if the maximum distance between any two CCHs with the same color is more than TTL hops, then there will be more than 1 color overlay for that color, and those overlays are not connected. Any lookup message flooded in one partial color overlay will not be forwarded to other partial overlays with the same color. This means a node may not be able to find all the copies of a file in the network.

In the simulations, we did color clustering using k=4 in 300-node networks and set TTL to values greater than 4. When k is 4, TTL has to be 5 or greater. Otherwise, broadcast messages will not be able to be forwarded to neighboring CCHs, which are at least 5 hops away. Figure 8(a) shows the percentage of connected color overlays for each TTL. In other words, it shows for each TTL, what is the probability to get all the CCHs with the same color connected. You can see that when TTL increases, the percentage also increases. The trend is true because more nodes will be touched with a larger TTL. But, the value of the percentage may vary in different networks because it depends on the diameter of the network. Figure 8(b) shows how many copies of the desired file are found for each TTL, no matter whether every CCHs with the same color is connected as a single overlay or not. It is easy to see that more copies can be found for a larger TTL. That is because there is a better chance

for more CCHs with the same color to get connected, thus lookup messages will be forwarded to more nodes.

5. Conclusions

This paper presents and evaluates XYZ, a generic peer-to-peer content location and routing system based on a self-organizing overlay network of nodes connected via the Internet. XYZ is partially centralized: some nodes such as CHs and CCHs play as system backbones, color overlay connectors, and local central indexes for files shared by local peers within the same color cluster. By using clusters and color overlays, messages can jump between nodes more quickly, traveling within color clusters and among color overlays. XYZ is fault-resilient, scalable, and reliably routes a lookup message to all the live nodes that have the file. You need at most $\lceil d/k \rceil + 2$ steps to route messages between any nodes, which means any lookup process can be done in a constant number of steps. Table 2 is the comparison of different P2P systems.

There are a couple of extensions that can make the XYZ protocol practical in actual systems. These extensions include caching and replication techniques for "Hot Spot" management. In reality, hot spots will be overloaded when many nodes request popular files from them in a short period of time. Some caching and replication techniques commonly applied to the World Wide Web are borrowed to solve this issue. Another interesting topic introduced by caching and replication is the question of storage space management. Because the storage space at each node is limited, and it will affect the performance of the system.

References

[1] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-min D-cluster formation in wireless ad hoc networks," In *Proceedings of INFOCOM 2000*, Mar. 2000.

[2] I. Stojmenovic, F. G. Nocetti, and J. S. Gonzalez, "Connectivity based *k*-hop clustering in wireless networks," *Telecommunication System,* 22:1-4, 205-220, 2003.

[3] C.R. Lin and M. Gerla, "Adaptive clustering for mobile wireless networks," *IEEE Journal on Selected Areas in Communications,* Vol.15, No. 7, pp. 1265-1275, Sept. 1997.

[4] M. Liu, R.R Talpade, A. McAuley, and E. Bommaiah, "Amroute: ad-hoc multicast routing protocol," *CSHCN T.R.*99-1 (*ISR T.R.* 99-8), Aug. 1999.

[5] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems", *Communications of the ACM*, Vol. 46, No. 2, pp. 43-48, Feb. 2003.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," In *Proceedings of ACM SIGCOMM,* Aug. 2001.

[7] I. Stoica, R. Morris , D. Karger , M. F. Kaashoek , and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," In *Proceedings of ACM SIGCOMM*, Aug. 2001.

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

[9] K. Hildrum, J. D. Kubatowicz, S. Rao, and B. Y. Zhao, "Distributed object location in a dynamic network," In *Proceedings of 14th ACM Symp. on Parallel Algorithms and Architectures,* Aug. 2002.

[10] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," In *Proceedings of ACM SPAA*, Jun. 1997.

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," In *Proceedings of ICSI Workshop on Design Issues in Anonymity and Unobservability*, Jun. 2000.

[12] P. Ganesan, Q. Sun, and H. Garcia-Molina, "YAPPERS: A peer-to-peer lookup service over arbitrary topology," In *Proceedings of INFOCOM 2003,* Apr. 2003.

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

[14] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing," *HPL-2002-57*, HP Laboratories Palo Alto, Mar. 2002.

[15] S. Androutsellis-Theotokis, "A survey of peer-to-peer file sharing technologies," *Athens University of Economics and Business*, Mar. 2003.

[16] A. Oram, Ed., "Peer-to-peer: harnessing the power of disruptive computation," *O'Reilly & Associates*, Mar. 2001.

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

[18] http://kazaa.com

[19] http://grokster.com

[20] http://morpheus.com

[21] Y. Liu, L. Xiao, and L. M. Ni, "Building a scalable bipartite P2P overlay network," In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Apr. 2004.

Table 2
Comparison of different P2P systems

P2P System	Algorithm Comparison Criteria				
	Model	Parameters	Hops to locate data	Routing state	Peers join and leave
XYZ	Partially centralized, Broadcast requests in overlay only	k-diameter of cluster c-number of colors/overlays d-diameter of network	$\lceil d/k \rceil + 2$	Constant	Joins: constant Leaves: $6k$
Gnutella	Broadcast request to as many peers as possible, download directly	None	No guarantee	Constant (approx 3-7)	Constant
CAN	Multidimensional coordinate space	N - number of peers in network d - number of dimensions	$d \cdot N^{1/d}$	$2 \cdot d$	$2 \cdot d$
Chord	Uni-dimensional, circular ID space	N - number of peers in network	$\log N$	$\log N$	$(\log N)^2$