An Efficient Hybrid Peer-to-Peer System for Distributed Data Sharing

Min Yang and Yuanyuan Yang

Department of Electrical & Computer Engineering, State University of New York, Stony Brook, NY 11794, USA

***Abstract*—Peer-to-peer overlay networks are widely used in distributed systems. Based on whether a regular topology is maintained among peers, peer-to-peer networks can be divided into two categories: structured peer-to-peer networks in which peers are connected by a regular topology, and unstructured peer-to-peer networks in which the topology is arbitrary. Structured peer-to-peer networks usually can provide efficient and accurate services but need to spend a lot of efforts in maintaining the regular topology. On the other hand, unstructured peer-to-peer networks are extremely resilient to the frequent peer joining and leaving but this is usually achieved at the expense of efficiency. The objective of this work is to design a hybrid peer-to-peer system for distributed data sharing which combines the advantages of both types of peer-to-peer networks and minimizes their disadvantages. The proposed hybrid peer-to-peer system is composed of two parts: the first part is a structured core network which forms the backbone of the hybrid system; the second part is multiple unstructured peer-to-peer networks each of which is attached to a node in the core network. The core structured network can narrow down the data lookup within a certain unstructured network accurately, while the unstructured networks provide a low cost mechanism for peers to join or leave the system freely. A data lookup operation first checks the local unstructured network and then the structured network. This two-tier hierarchy can decouple the flexibility of the system from the efficiency of the system. Our simulation results demonstrate that the hybrid peer-to-peer system can utilize both the efficiency of structured peer-to-peer network and the flexibility of the unstructured peer-to-peer network and achieve a good balance between the two types of networks.**

Keywords: Peer-to-peer systems, P2P, structured peer-to-peer, unstructured peer-to-peer, hybrid, overlay networks.

I. Introduction

Last few years have witnessed a rapid development of peer-to-peer networks. Research has shown that a large fraction of traffic in the Internet is occupied by peer-to-peer applications [2]. A peer-to-peer (P2P for short) network is a logical overlay network on top of a physical network. Each peer corresponds to a node in the peer-to-peer network and is resided in a node (host) in the physical network. All peers are of equal roles. The links between peers are logical links, each of which corresponds to a physical path in the physical network. The physical path is determined by a routing algorithm and composed of one or more physical links. Logical links can be added to the peer-to-peer network arbitrarily as long as a corresponding physical path can be found, that is, the physical network is connected.

The flexibility of the overlay topology and the decentralized control of the peer-to-peer network make it suitable for distributed applications. For example, it can be used for distributed data (file) sharing, where peers announce the data (files) they have and exchange data (files) from each other through a loosely formed peer-to-peer network, or for collaborative web caching in which web pages are cached in collaborative peers to reduce network delay for URL requests, or for application layer multicast in which peers are group members and the peer-to-peer overlay network is a multicast tree. It can also be used for distributed computing which utilizes the idle resources in the network for a huge computing task. Finally, it can be used to provide communication anonymity in which the sender's identity is concealed.

Based on whether a regular topology is maintained among peers, peer-to-peer networks can be divided into two categories: structured peer-to-peer networks in which peers are connected by a regular topology, and unstructured peer-to-peer networks in which the network topology is arbitrary. Structured peer-to-peer networks build a distributed hash table (DHT) on top of the overlay network. The hash table supports efficient data insertion and lookup. Given a key of the data item, the corresponding value of the data item can be inserted or found by transforming the key to a hash value by a hash function. The hash value is the index of the data item and all the hash values form the key space. In DHT, the key space is divided among peers. Each peer is responsible for one partition of the key space. Peers are connected by an overlay network through which the requests of data insertion and lookup are delivered. Structured peer-to-peer networks can provide efficient and accurate query service but need a lot of efforts to maintain the DHT, which makes it vulnerable to frequent peer joining and leaving, also known as *churn*. Unstructured peer-to-peer networks organize peers into an arbitrary network topology, and use flooding or random walks to look up data items. Each peer receiving the flooding packets or random walk packets checks its own database for the data item queried. This approach does not impose any constraint on the network topology. It can perform complex data lookup and support peer heterogeneity. Unstructured peer-to-peer networks are resilient to churn while they usually achieve this goal by sacrificing the data query efficiency and accuracy.

Hence, neither structured peer-to-peer networks nor unstructured peer-to-peer networks can provide efficient, flexible and robust service alone. In this paper, we propose a hybrid peer-to-peer system for distributed data sharing which combines both types of peer-to-peer networks. In the proposed hybrid system, a structured ring based core network forms the backbone of the system and multiple unstructured peer-to-peer networks are attached to the backbone and communicate with each other through the backbone. Data is generated and distributed among the peers. The core structured network provides an accurate way to narrow down the queried data within a certain unstructured network, while the unstructured networks provide a low cost mechanism for peers to join or leave the system freely. Our simulation results show that the hybrid peer-to-peer system can utilize both the efficiency of the structured peer-to-peer network and the flexibility of the unstructured peer-to-peer network, and achieve a good balance between the efficiency and flexibility.

The rest of paper is organized as follows. In Section II, we briefly describe some related work. We present the new hybrid peer-to-peer system in Section III. We give some enhancements to the hybrid peer-to-peer system in Section IV. In Section V, we present the simulation results and discuss the performance of the system. Section VI gives the concluding remark and future

The research was supported in part by the U.S. National Science Foundation under grant numbers CCR-0207999 and CCF-0744234.

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

work.

II. Related Work

Many peer-to-peer networks have been proposed for different applications in the literature, see, for example, [1]-[15]. In this paper we focus on peer-to-peer networks for efficient distributed data (file) sharing among peers.

The Content Addressable Network (CAN) [10] was proposed to provide a scalable indexing mechanism for file sharing over a large network. As a distributed infrastructure, CAN provides hash table-like functionality over Internet-like scales. Both peers and data are hashed to a virtual d-dimensional Cartesian coordinate space. The entire space is partitioned to distinct zones such that each peer is in charge of one zone. Every peer maintains a routing table which holds the IP address of its neighbors in the coordinate space. The data is stored in and retrieved from the peer that owns the zone covering the data. CAN takes advantage of the ordering of the Cartesian coordinate space in the routing algorithm. Packets are forwarded along the straight line connecting the source and the destination in the Cartesian coordinate space. When a new peer joins the system, some existing zone will be split into two zones one of which is assigned to the new peer, and all the related peers need to update their neighbor lists. When a peer leaves the system, a neighboring peer will take over the zone by running a takeover algorithm, and all the related peers need to update their neighbor lists again.

Chord [11] organizes the peers into a circle which is called a chord ring, where each peer is assigned an ID. Peers are inserted into the ring in the order of their IDs. Each peer has two neighbors: successor and predecessor. When a peer joins the system, it first finds the position to insert the new peer. Then the successor pointers of both the new peer and an existing peer must be changed. The correctness of Chord relies on the fact that each peer is aware of its successor. To guarantee this, each peer maintains a successor list of size r which contains the peer's first r successors. Each data item also has an ID and is stored in a peer such that the ID of the data item is between the ID of the peer and its predecessor. Packets are forwarded along the circle. In order to accelerate the search, each peer maintains a finger table, where each finger points to a peer with a certain distance from the current peer. Chord uses a "stabilization" protocol running in the background to update the successor pointers and finger tables. Compared to CAN, Chord is simpler as the key is hashed in a one-dimensional space.

Gnutella [13] is a decentralized unstructured peer-to-peer network. The network is formed by peers joining the network following some loose rules. There is no constraint on the network topology. To look up a data item, a peer sends a flooding query request to all neighbors within a certain radius. As Gnutella has no requirement on the network topology and data placement, it is extremely resilient to peer joining and leaving the system frequently. However, flooding is not scalable and consumes a lot of network bandwidth.

BitTorrent [14] is a centralized unstructured peer-to-peer network for file sharing. A central server called tracker keeps track of all peers who have the file. Each file has a corresponding torrent file stored in the tracker which contains the information about the file, such as its length, name and hashing information. When receiving a download request, the tracker sends back a random list of peers which are downloading the same file. When a peer has received the complete file, it should stay in the system for other peers to download at least one copy of the file from it. Since BitTorrent uses a central server to store all the information about the file and the peers downloading the file, it suffers so called "single point of failure" problem which means that if the central server fails, the entire system is brought to a halt.

YAPPERS [15] combines both structured peer-to-peer networks and unstructured peer-to-peer networks to provide a scalable lookup service over an arbitrary topology. Each peer has an immediate neighborhood (IN) which contains all peers within h hops of the peer and an extended neighborhood (EN) which contains all peers within $2h + 1$ hops of the peer. Both data keys and peers are hashed to different buckets or colors. Data is stored in the peer in the same color. For a data lookup, a peer first checks the peers in the IN in the same color, then these nodes will forward the request to the peers in their IN, and so on. Finally, all the peers in the same color will be checked. However, YAPPERS is designed for efficient partial lookup which only returns partial values of data. For a total lookup, YAPPERS still needs to flood the request to all peers which are in the same color as the data.

III. The Hybrid Peer-to-Peer System

In this section, we first give an overview of the new hybrid peer-to-peer system we propose, in which a tunable system parameter p_s is defined to characterize the system performance. Then we describe how to maintain the peer-to-peer network topology when peers join and leave the system. Finally, we describe how to insert and look up data items in the system.

A. System overview

The new hybrid peer-to-peer system is composed of two parts: a core transit network and many stub networks each of which is attached to a node in the core transit network. The core transit network, called *t-network*, is a structured peer-to-peer network which organizes peers into a ring similar to a chord ring. We call peers in the t-network *t-peers*. Each t-peer is assigned a peer ID (p_id), which is a positive integer. Peers are inserted to the ring in the order of their p_ids. Each t-peer maintains two pointers which point to its successor and predecessor respectively. A finger table is also used to accelerate the search. A stub network, called *s-network*, is a Gnutella-style unstructured peer-to-peer network. We call the peers in an s-network *s-peers* except for the t-peer attached to this s-network. The topology of an s-network is arbitrarily formed. Each s-network is attached to a t-peer and this t-peer belongs to both the t-network and the s-network. Fig. 1 shows the overview of the hybrid peer-to-peer system.

We define a system parameter p_s as the proportion of s-peers in the system. p_s is a tunable system parameter which has a great impact on the system performance. In particular, when p_s equals to 0, the system degenerates to a ring-based structured peer-to-peer network; when p_s equals to 1, the system becomes a Gnutella-style unstructured peer-to-peer system; and when p_s equals to 0.5, a half of the peers in the system are t-peers while

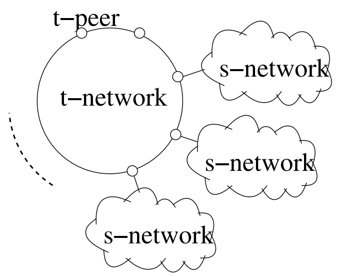


Fig. 1. Overview of the proposed hybrid peer-to-peer system.

the other half of the peers are s-peers. By tuning the system parameter, we can observe how the system parameter affects the system performance so that we can select an optimal value for p_s to maximize the performance in different applications.

The basic idea behind the hybrid peer-to-peer system is that the t-network is used to provide efficient and accurate service while the s-network is used to provide approximated best-effort service to accommodate flexibility. Peers can join either t-network or s-network directly. An s-network is composed of peers that serve the data of some common properties. A data lookup is confined within an s-network if the queried data has the common properties served by the s-network. The lookup request is passed around the s-network through flooding or random walk. Although flooding may generate a lot of network traffic, it can greatly simplify peer joining and leaving process, which makes the system robust to churn. On the other hand, since the s-network contains only a small proportion of the total number of peers, flooding is confined within a small number of peers. When the queried data is served by another s-network, the data query request is first forwarded to the t-network through the t-peer in the s-network generating the request. In the t-network, the request will be forwarded along the ring until it reaches the s-network serving the queried data. In the s-network the request will be delivered to the s-peers by flooding again. The t-network links all the s-networks together and provides an efficient way to locate the desired s-network. The stableness of the t-network is critical to the system performance because all the communications between different s-networks are through the t-network. As a structured peer-to-peer network, the t-network is vulnerable to churn mostly because the t-network needs to recalculate the pointers in the finger tables whenever a t-peer joins or leaves. However, the hybrid system can reduce the topology maintenance overhead caused by peer joining or leaving. On one hand, a large portion of peers join the s-networks directly without disturbing the t-network; on the other hand, an s-peer can be selected to substitute the leaving t-peer in the same s-network, i.e. the selected s-peer will become a t-peer. In this case the total number of t-peers is unchanged. Therefore, there is no need to recalculate the pointers in the finger tables, and only a simple update is needed.

In this paper, we focus on applying the hybrid peer-to-peer system to distributed data sharing. A data item is represented by a (*key*, *value*) pair. A *key* is a label or name of the data, such as a file name, while a *value* is the content associated with the key, such as a file. A peer uses operation store (*key*, *value*) to insert the data item into the system and operation lookup (*key*) to obtain the value of the data item. Before performing the store or lookup operations, a peer hashes the data key to an integer d_id which is in the same range as p_id. As mentioned earlier, s-peers are grouped into different s-networks such that each s-network serves the data of some common properties. In the hybrid system, the p_ids of t-peers divide the range of the d_id into several segments. Each s-network is responsible for the data whose d_ids lie in the same segment. Both store and lookup operations try the local s-network first if the data item is served by the local s-network, otherwise turn to the t-network. Thus, such two-tier hierarchy structure can provide efficient lookup in the top tier while maintaining the flexibility in the bottom tier.

B. Peer join/leave

Peer-to-peer networks are highly dynamic and autonomic systems in which peers can join or leave the systems at any time. Peers that want to join the system first contact a well-known server to obtain an arbitrary existing peer in the system. The IP address of the server can be acquired by a DNS-like public service. Unstructured peer-to-peer networks process join or leave requests in a more flexible way than structured peer-to-peer networks largely because structured peer-to-peer networks have to maintain the network topology after peers join or leave. Since the hybrid peer-to-peer system contains both structure peer-to-peer network (t-network) and unstructured peer-to-peer network (s-network), the operations of peer joining or leaving the system are based on which network they belong to. Next we will discuss these two types of networks separately.

B.1 T-peer join/leave

The join operations of t-peers are similar to that in the Chord [11]. Each t-peer maintains two pointers that point to its successor and predecessor along the ring respectively. After the joining peer obtains the arbitrary t-peer, it sends a join request containing its p_id to the t-peer. This join request will be forwarded along the ring until it reaches the t-peer such that the p_id of the joining peer is between the p_id of the t-peer and its successor. The t-peer will initiate a join process and the joining peer is inserted between the t-peer and its successor.

The p_id of a new peer is generated at the server. The server has several options to generate the p_id. One way is to generate the p_id by hashing the IP address of the new peer. Another way is to generate the p_id based on the location of the new peer. This can make the peers close to each other in the physical network also close to each other in the overlay network. Moreover, the server can generate a random p_id for the new peer. However, the p_id generation process does not guarantee the uniqueness of p_id. In the case of a conflict, the t-peer initiating the join process will generate a new p_id which lies in between the p_id of itself and its successor. The new p_id can be random or simply the midpoint for load balancing purpose.

After the join process completes, the segment of the id space represented by the successor has changed. The peers in the successor's corresponding s-network should transfer part of its data load to the new peer, which is referred to as load transfer. The peers checks the data items it stores and transfers all the data items whose d_id lie between the p_id of the new peer and the predecessor of the new peer.

In a structured peer-to-peer network such as the Chord, when a peer leaves the system, all the pointers related to the leaving peer must be updated. These pointers include the successor pointer in its predecessor, the predecessor pointer in its successor and the finger pointers in all the peers on the Chord ring. The hybrid peer-to-peer system can reduce some update work as it does not need to recalculate new finger pointers. When a t-peer wants to leave the system, it transfers its role to one of the s-peers in its s-network. A selected s-peer will change its role from an s-peer to a t-peer and take over the neighbors and the pointers of the original t-peer. By substituting the t-peer with an s-peer, the number and position of t-peers remain the same. Other t-peers only need to substitute the leaving t-peer with the new t-peer in the finger table.

Table 1 lists the pseudo-code for join and leave operations.

B.2 S-peer join/leave

Each s-peer belongs to an s-network and maintains a list of its neighbors. A neighbor can be either an s-peer or a t-peer. After an s-peer acquires the IP address of the random peer, it adds the peer to its neighbor list. Then it notifies the random peer to add itself to its neighbor list and the join operation is completed.

In a Gnutella-style peer-to-peer network, the data lookup is through flooding. The range of flooding is determined by the search radius, that is, the TTL (time-to-live) value of the packet. As the data is distributed around the network randomly, the search radius is critical to the probability of finding the desired data item. For the same topology and the same peer that initiates the search, the longer the search radius, the higher the probability of finding the desired data item, but the longer the latency required. Note that if we add some simple constraints on choosing the random peer when a new peer joins, we may shorten the network diameter and thus reduce the search radius without sacrificing the success ratio of finding the desired data item. Next we will discuss how to add such constraints.

First, we restrict the random peer to be picked to only t-peers. Thus, all s-peers are connected with one t-peer. As a result, the diameter of an s-network is at most 2, and one data lookup can reach all the peers within 2 hops. The topology of the s-network is a star centered at a t-peer. Although the data lookup can achieve short latency in such an s-network, there is a notable disadvantage that the load is extremely unbalanced. The t-peer maintains a long neighbor list while the s-peer has a neighbor list with only 1 neighbor. Each data lookup request has to be forwarded through the t-peer. In order to alleviate this problem, we put another restriction on the degree of peers. When the degree of a peer reaches a threshold δ, it passes the join request to one of its neighbors randomly. The join request will be passed until it arrives a peer whose degree is less than δ. In our simulation, we use this scheme for s-peer joining. The new s-peer searches from a t-peer along a random branch until it finds a peer with degree less than δ. This peer is called the *connect point (cp)* of the new s-peer. Besides the neighbor list, each s-peer maintains two pointers that store the address of the t-peer of the s-network and its cp.

When an s-peer leaves the system, it should notify all its neighbors about the leaving. The neighbors then delete it from their neighbor lists. The neighbor whose cp is the leaving peer should rejoin the s-network by sending a join request to the t-peer again. The leaving s-peer should also choose a neighbor to transfer the load to.

TABLE 1
JOIN/LEAVE ALGORITHM FOR T-PEERS

```
//Inserting new peer $n$ between peer $pre$ and peer $suc$
pre.join(n):
  //peer $pre$ checks if $id$s conflict
  pre.check(n.id); //n.id is the $p\_id$ of peer $n$
  //peer $pre$ sets its successor pointer to peer $n$
  pre.successor = n;
  //peer $n$ sets its predecessor pointer to peer $pre$
  n.predecessor = pre;
  //peer $n$ sets its successor pointer to peer $suc$
  n.successor = suc;
  //peer $suc$ sets its predecessor pointer to peer $n$
  suc.predecessor = n;
  //peer $suc$ transfers part of its data load to peer $n$
  suc.loadtransfer(n.id);

n.leave():
  //peer $pre$ sets its successor pointer to peer $suc$
  pre.successor = suc;
  //peer $suc$ sets its predecessor pointer to peer $pre$
  suc.predecessor = pre;
  //peer $n$ transfers all of its data load to peer $suc$
  foreach peer m inn.loaddump();

pre.check(n.id):
  if(n.id==id) //id is the $p\_id$ of peer itself
    //if $id$s conflict, assign a new $id$ to peer $n$
    n.id = (id+suc.id)/2;

suc.loadtrandfer(n.id):
  foreach peer in the current s-network
    foreach data in database
      if(data.id<=n.id) //data.id is the $d\_id$ of data
        //insert the data item for which peer $n$ is
        //responsible into peer $n$
        n.insert(data);
        //delete the data item from peer $suc$
        suc.delete(data);

n.loaddump():
  //move all the data items from peer $n$ to peer $suc$
  foreach data in database
    suc.insert(data);
    n.delete(data);
```

Also, each s-peer periodically broadcasts “Hello” messages to all its neighbors to detect crashes or link failures. If a peer crashes, the load on the peer is lost. In the simulation, we simulate this scenario and observe its impact on the data lookup failure ratio.

C. *Concurrent join/leave*

Peer-to-peer networks are highly dynamic systems since peers usually are end hosts that are in charge of different individuals or groups. Concurrent joins and leaves are very common and can greatly degrade the performance if not handled carefully. In this paper, we assume that all peers leave the system friendly, that is, no sudden crashed peers.

The concurrency handling in the s-network is simpler than that in the t-network. When an existing peer receives two join requests, it follows the FCFS (First Come First Serve) rule, that is, the second join request will be passed to the next neighbor if the degree of the peer reaches the limit after receiving the first join request. When two or more s-peers leave the system simultaneously, no special action is needed because the disconnected parts will rejoin the s-network as described in the previous subsection.

For the t-network, concurrency handling is much more complicated as it involves three t-peers and the topology constraint is strict. The concurrent joins or leaves may lead to an incorrect topology or break the t-network apart if not handled carefully. For example, suppose a t-peer is inserting a new peer, say, x, between itself and its successor. After setting its successor pointer to the new peer, it receives another join request indicating that another new peer, say, y, should be inserted between x and its successor. Thus, the t-peer will pass the join request to its new successor x. However, the join operation of x is not completed, and the successor and predecessor pointers in x are not set correctly. Therefore, the join request of y will not be handled correctly.

We adopt the idea of the concurrency handling in the database system for the concurrent joins or leaves in the t-network. The join and leave requests are sequentialized such that the next request is not processed until the previous request finishes. For a join request, it follows a join triangle as shown at the left of Fig. 2. When peer pre receives the join request of the new peer, it sets a mutex variable $joining$ that indicates some peer is being inserted between peer pre and peer suc. Now peer pre will not accept any leave requests including that from itself. If a new join request comes before the previous request is completed, peer pre will insert the new request to a queue and process the request queue after the previous join request finishes. Then peer pre sends a packet including the address of peer suc to the new peer. The new peer sets its successor and predecessor pointers to suc and pre respectively and sends another packet to peer suc. After receiving the packet, peer suc updates its predecessor pointer and sends a packet back to peer pre. When peer pre receives this packet, it sets its successor pointer to the new peer and continues to process the next join request in the queue. If the queue is empty, it resets the mutex variable.

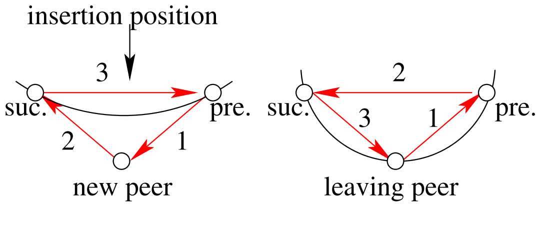


Fig. 2. Concurrent join/leave operation for t-peers.

When a peer leaves the system, it follows the leave triangle which is shown on the right of Fig. 2. When a peer is leaving, it also sets a mutex variable $leaving$. Now the peer will not accept any new join request (if the join request queue is not empty, the peer should process the join request first) and leaving request. Then the leaving peer sends a packet to peer pre including the address of peer suc. Peer suc sets its successor pointer to peer suc and sends a packet to peer suc including the address of the leaving peer. After receiving the packet, peer suc will check whether the leaving peer included in the packet is what its predecessor pointer is pointing to. Only if they are the same peer, will the peer suc sets its predecessor pointer to peer pre and sends a packet to the leaving peer to notify the completion of the leaving operation.

D. *Data insertion/lookup*

Data is generated and inserted to the system by peers. As mentioned earlier, each s-network is responsible for a range of ID space. The peer generating the data item first hashes the key into this space. If the d_id lies in the range of the current s-network, the data item is inserted to its database and the data insertion is completed. If the d_id does not lie in the range, the data item is sent to the t-peer of the current s-network. Then it is forwarded along the t-network until it reaches the t-peer in charge of the ID range covering d_id. Then the data item is inserted into the database of the t-peer.

Note that although this data placement scheme is simple and easy to implement, the data load may be imbalanced. Since the data generated in other s-networks will always be stored in the t-peer, the data load in t-peers is much heavier than that in s-peers. We can improve this by spreading the data load to the neighbors of the t-peer. When a t-peer receives a data insertion request, it picks a random peer from its neighbors and itself and then sends a data insertion request to the random peer. The random peer will do the same until the data item is finally inserted to the system. In the performance evaluation section, we implement both data placement schemes and study their impact on the probability density function of the number of data items per peer.

When a peer looks up a specified data item, it first obtains the d_id of the data item by hashing the data key. If the d_id lies in the current s-network, the peer floods lookup packets around the s-network and sets a timer for it. The timer will be reset if the peer receives the data item or expire, which indicates that the data item is not found. The peer may choose to increase the TTL value and the expiration duration of the timer and reflood the lookup packets. If the d_id does not lie in the current s-network, the peer sends a lookup request to the t-peer and also sets a timer for it. Similar to data insertion, the data lookup request is forwarded along the t-network until it arrives at a proper t-peer which will then flood data lookup packets around its s-network. Each peer receiving the lookup request will check its database for data item d_id. If the data item is found in its database, the peer will stop flooding and send the data item to the peer requesting the data item directly.

IV. Enhancements

In this section, we discuss some enhancements which can make the hybrid peer-to-peer system more flexible and improve the system performance when applied to different applications.

A. Supporting link heterogeneity

Link heterogeneity is a common phenomenon in networks. In peer-to-peer networks, link heterogeneity refers to the fact that peers have different access link capability. In general, common access links include dial-up connections, ADSL and high speed cable, and the ratio of the link capacity of the fastest link to the slowest link can be more than 1000.

Due to the imbalance of link capacity, if a peer with high link capacity is receiving messages from a peer with low link capacity, its download speed is upper bounded by the download speed of the low link capacity peer. In other words, the bandwidth of the high link capacity peer is wasted. To maximize the usage of all the link capacities, we connect the peer with higher link capacity in the system to multiple peers with lower link capacities so that the fast peer can download or upload data to the multiple slow peers simultaneously.

As mentioned earlier, the t-peers carry more traffic than the s-peers. Even after we add some restriction to the connect point selection, the load imbalance cannot be removed completely. On the contrary, we can make use of this imbalance for link heterogeneity consideration. Since t-peers are connected to more other peers than s-peers on average, we assign peers with higher link capacities as t-peers while peers with lower link capacities as s-peers. In practice, each peer attaches its link capacity value to the packet sent to the server. Based on the value, the server decides whether the peer is a t-peer or an s-peer.

The link heterogeneity idea can be applied to the s-network construction as well. When an s-peer joins an s-network, the connect point first checks its *link usage* which is defined as the ratio of the degree to the link capacity of the peer. If the link usage is low enough, the connect point will establish a link between itself and the joining s-peer. If the link usage is high, the connect point will notify the joining s-peer and choose a random branch for it as discussed in Section III-B.2.

B. Supporting topology awareness

Since overlay networks are logical networks on top of physical networks, the overlay links are logical links. Each logic link is composed of one or more physical links. The overlay links are added arbitrarily as needed. As a result, the topology of the overlay network may be different from the topology of the physical network. Two nodes which are close to each other in the overlay network may be far away in the physical network. Such topology mismatch may increase the *link stress* and degrade the performance, where link stress is defined as the number of copies of a message transmitted over a certain physical link.

One way to solve this mismatch problem is to cluster the peers such that the peers in the same cluster are close to each other with respect to the latency. We can adopt the idea of binning scheme introduced in [16] to construct a topology aware overlay network. In the scheme, the well-known server is responsible for choosing some peers as landmarks. Each new peer receives the list of landmarks from the server as a response of its join request. The new peer then sends probe messages to the landmarks to learn the distances between them and itself. The landmark peers are listed in an ascending order of distances. The ordered list acts as the coordinate of the new peer in the system. The coordinate is sent to the server, then the server assigns the new peer a cluster ID based on its coordinate. Peers with the same coordinates form a cluster. Intuitively, peers in the same cluster are close to each other. The server assigns peers in the same cluster to the same s-network. In the case that the number of s-networks is more than the number of clusters, the peers in the same cluster may be assigned to different s-networks. The peers are assigned to several s-networks in a round robin manner such that the numbers of s-peers of different s-networks are balanced.

In addition, every two landmark peers should not be too close to each other. A new peer cannot be a landmark if its coordinate is the same as one of the existing landmark peers. A landmark peer is removed from the landmark list if it has the same coordinate as another landmark peer.

C. Interest based s-networks

In previous sections, we assigned the new s-peer randomly to existing s-networks. However, the assignment may be determined by applications. For example, the s-network may be composed of peers that are interested in the same type of data. The data generated in one s-network is looked up mostly by a peer in the same s-network. In this case, a new s-peer can indicate its interest when sending the join request to the server. The server will acknowledge with a t-peer whose s-network matches the interest, and the new s-peer will join that s-network. With the application-specific assignment, s-peers are more likely to generate data stored in its own database and look up data in its own s-network. Hence, the latencies of both data insertion and lookup can be reduced. This way, most of traffic related to data operations is confined within one s-network and the burden of the t-network is alleviated.

D. Bypass link

As discussed in the previous subsection, an s-network can be composed of s-peers with the same interest. However, in some applications, an s-peer may be interested in several types of data at the same time. In the hybrid peer-to-peer system, an s-peer can only belong to one s-network at a time. Therefore, we add random bypass links between different s-networks to accelerate the data operations.

Another reason to use bypass links is to alleviate the burden of the t-network. Since the t-network is responsible for transmitting join or leave requests of t-peers and performing all the data operations among s-networks, the t-network links bear much heavier load than that in s-networks. With bypass links, part of the data operations can be diverted from the t-network to the bypass links.

Bypass links can be added according to the following three rules: first, a bypass link cannot be added to a peer unless the degree of the peer is less than the threshold δ; second, if peer a generates and inserts a data item in peer b, and peer a and peer b are in different s-networks, a bypass link should be added between peer a and peer b; third, if peer a looks up and finds a data item in peer b, and peer a and peer b are in different s-networks, a bypass link should be added between peer a and peer b.

Peers maintain bypass links in a loose way. Each bypass link is attached with a timer. When the timer expires, the bypass link is deleted. Transmitting a packet through the bypass link will refresh the attached timer. When a peer leaves the system, the bypass links it maintains are lost.

E. BitTorrent-style s-network

Although the hybrid peer-to-peer system uses the Gnutella network as the prototype of the s-network, it can also easily deploy the BitTorrent-style s-network. In a BitTorrent-style s-network, the t-peer acts as the tracker. When a data item is inserted to the database of an s-peer, it is reported to the t-peer. The t-peer maintains all the information of data items stored in its s-network. A data lookup request is sent to the t-peer directly. T-peer responds the data lookup with the peer which has the data item. Then data item is delivered between the two peers directly. In a BitTorrent-style s-network, the t-peer plays a more important role than that in Gnutella-style s-network and no flooding is needed.

V. Simulation Results

In this section, we evaluate the performance of the proposed hybrid peer-to-peer system through simulations. We have implemented the system on the NS2 simulator. The network topologies used in the simulations are random transit-stub network topologies generated by GT-ITM software [19]. Each network topology is composed of 1000 nodes, and each node is assigned to be either an s-peer or a t-peer randomly. The ratio between the number of s-peers and the total number of peers is determined by the system parameter p_s. Note that when $p_s = 0$, the hybrid system degenerates to a ring-based structured peer-to-peer network. When $p_s = 1$, the hybrid system becomes a Gnutella-style unstructured peer-to-peer system. In the simulations, we let p_s increase from 0 to 1 so that we can compare the hybrid system with both the structured peer-to-peer network and the unstructured peer-to-peer network. We set the peers with heterogeneous link capacities such that $1/3$ of the peers have the highest link capacities, $1/3$ of them have the lowest link capacities and $1/3$ of them have the medium link capacities. The highest link capacity is 10 times of the lowest link capacity. The landmarks are predetermined so that they are uniformly distributed around the network. We will first compare the data distribution under two different data placement schemes discussed in Section III-D, and then compare the data lookup failure ratio and data lookup efficiency by tuning the system parameter p_s.

A. Data distribution

As discussed in Section III-D, when a data item is generated in an s-network and stored in another s-network, it will be forwarded along the t-network until it arrives at a proper t-peer. There are two options for the t-peer: insert the data item to its own database or insert it to the database of a random neighbor. The first option will cause imbalanced data load among peers while the second option can greatly alleviate the imbalance.

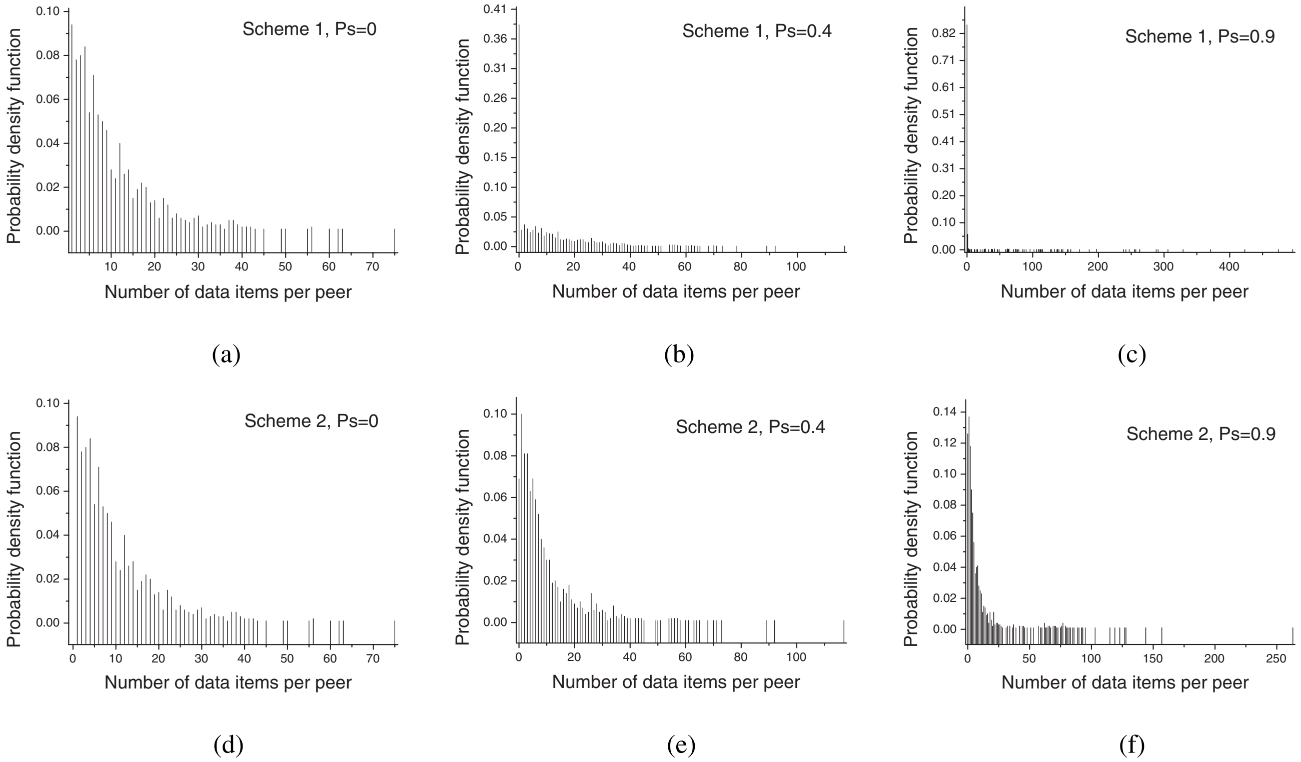
Fig. 3 shows the different probability density functions of the number of data items per peer under the two schemes. In order to display the curves in the center of the figure, the y axis starts from -0.01 and both x axis and y axis are rescaled. Fig. 3(a)-(c) are the probability density functions for the first data placement scheme when p_s is equal to 0, 0.4 and 0.9, respectively. We can see that when $p_s = 0$, more than 50% of the peers store less than 10 data items, and the rest of the peers store data items ranging from 10 to 80. When p_s increases to 0.4, the number of peers without any data occupy 38% of the total peers, and the rest of the peers store data items ranging from 0 to 120. When $p_s = 0.9$, the proportion of peers without any data item is 85%, and the highest number of data item per peer is more than 500. The peers are evenly distributed along the x axis. We can observe that with the increase of p_s, the data items are more likely to be stored in t-peers. The data items generated by an s-peer is finally inserted in some t-peer after traveling along the t-network. Most of s-peers have no data items stored. Fig. 3(d)-(f) show the probability density functions for the second data placement scheme when p_s is equal to 0, 0.4 and 0.9 respectively. The number of peers without any data item drops dramatically. In Fig. 3(f), the number of peers with no data item is only around 12% compared to 85% in Fig. 3(e), and 50% of the peers store data items less than 20. We can draw the following conclusion from the figures: when p_s is small, the two schemes can distribute the data items evenly among the peers; when p_s is large, the first scheme stores most of the data items in a few peers while the second scheme can balance the data distribution so that each peer stores a similar amount of data items.

B. Lookup failure ratio

Lookup failure ratio is defined as the the number of failed data lookups divided by the total number of data lookups. Here we assume that the data is inserted to the system before it is looked up by some peer.

The lookup failure ratio is affected by the TTL value, the diameter of the network and the position of the peer initiating the data lookup. Here we compare the lookup failure ratio of the hybrid peer-to-peer system by tuning both the TTL and p_s.

Fig. 4(a) shows the lookup failure ratios under different TTL values when p_s increases from 0 to 1. As illustrated in the figure, structured peer-to-peer networks achieve zero lookup failure ratio, while unstructured peer-to-peer networks have a greater than zero lookup failure ratio depending on the TTL value. When $p_s < 0.5$, the lookup failure ratio of the hybrid system is around 0 for all TTL values. Since the number of s-peers is less than the number of t-peers, an s-network has less than one s-peer on the average. Thus the flooding packets from a t-peer can reach all the s-peers within one hop on the average. When p_s increases, the lookup failure ratio increases exponentially. When TTL = 1 and $p_s = 0.9$, the lookup failure ratio is increased to 18%. From the figure we can also observe that increasing TTL value can reduce the lookup failure ratio dramatically. When $p_s = 0.9$, the lookup failure ratio is 4% if TTL = 4, compared to 14% if TTL = 2 and 18% if TTL = 1. If we set p_s to less than 0.5, the lookup failure ratio is almost zero



(a) (b) (c)

(d) (e) (f)

Fig. 3. Probability density functions of the number of data items per peer under different p_s values for the two data placement schemes.

which is the same as structured peer-to-peer networks. On one hand, increasing p_s will increase the lookup failure ratio; on the other hand, increasing p_s can increase the lookup efficiency by reducing the lookup latency and the number of peers a lookup may contact. We will discuss this tradeoff later and deduce the optimal p_s value.

We now consider the impact of peer crash on the lookup failure ratio. Peer crash means that the peer has no time to transfer the data load to one of its neighbors. In the simulation, the peers are chosen randomly to leave the system without transferring its data load. Fig. 4(b) shows the lookup failure ratio when different proportions of peers leave the system. We can see that the lookup failure ratio increases linearly to the number of peers leaving the system. Since as more peers are leaving the system, the data stored in the peers is no longer available. With the increase of p_s, the lookup failure ratio remains at the same level. This again shows that the improved data placement scheme can distribute the data load among peers evenly regardless of the value of p_s.

Generally speaking, higher p_s value causes higher lookup failure ratio. Increasing TTL value can alleviate the situation greatly. When peers crash, the increase of the lookup failure ratio is proportional to the number of crashed peers and changing p_s has no impact on the lookup failure ratio.

C. Lookup efficiency

In an ideal system, a high lookup success ratio should not sacrifice the lookup efficiency. We evaluate the lookup efficiency in terms of time and space. For the time, the lookup efficiency is simply the lookup latency defined earlier. For the space, it is the bandwidth involved in the lookup. We approximate the bandwidth by the number of peers a lookup may contact, denoted as $connum$. The less the $connum$ is, the less the bandwidth is needed.

Fig. 5(a) shows the average lookup latency with and without link heterogeneity consideration under different p_s values. As the figure shows, structured peer-to-peer networks have larger lookup latency than unstructured peer-to-peer networks while a hybrid system has a lookup latency in between. While p_s increases, the average lookup latency decreases more quickly when p_s is small. This is due to the fact that the number of t-peers decreases when p_s increases. A typical data lookup is composed of three steps: an s-peer sends data lookup request to the t-peer; the lookup request is forwarded along the t-network until it reaches a proper t-peer; the lookup request is flooded around the s-network. As both the first and third steps are confined by the TTL value in the lookup packet, the number of hops a lookup experiences is determined by the second step, which is proportional to the total number of t-peers. When p_s is large, the average size of s-networks is large which results in a larger search radius and lookup latency. This latency increase partially offsets the latency decrease so that the slope of the curve is more gradual when p_s is large.

When link heterogeneity is considered, the average lookup latency is further decreased. When p_s is between 0.4 and 0.8, this latency decrease is more obvious. The lookup latency is decreased by around 20% when $p_s = 0.7$.

Fig. 5(b) shows the average lookup latency with and with-

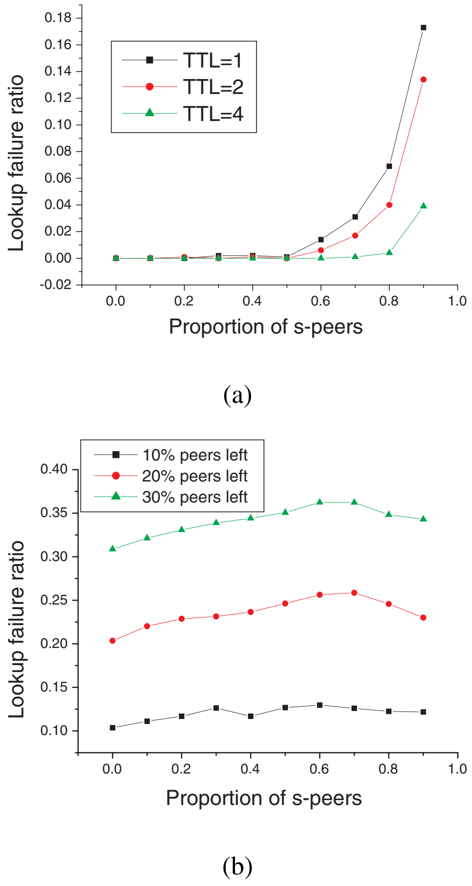


Fig. 4. (a) Lookup failure ratio under different TTL values. (b) Lookup failure ratio when peers crash under different p_s values.

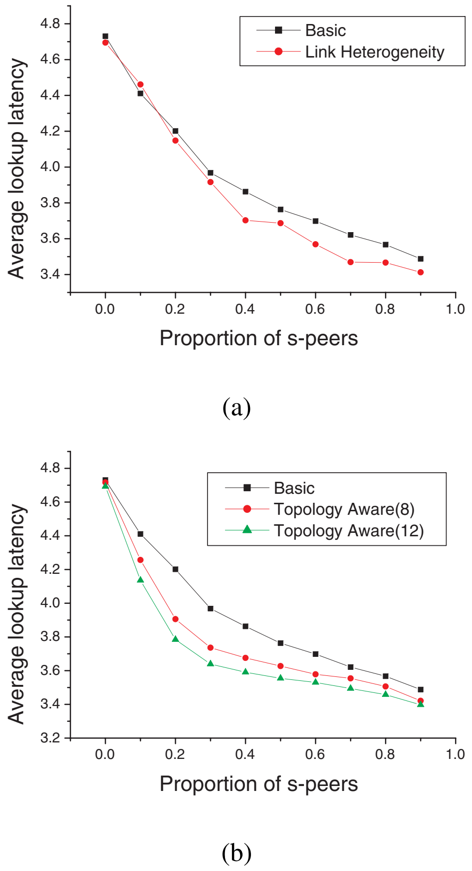


Fig. 5. Average lookup latency under different p_s values.

out topology awareness consideration under different p_s values. We consider two possible numbers of the landmarks: 8 and 12. From the figure we can see the lookup latencies are the same when $p_s = 0$. When p_s increases, the latency of topology awareness scheme decreases much faster than the basic scheme. The more the landmarks, the less the lookup latency. When p_s approaches 0.9, the three curves tend to merge. This means the topology awareness has little impact when the number of the s-networks is small. Because in this case, the peers close to each other are assigned to the same s-network with high probability.

It can be seen that extending the basic scheme to support link heterogeneity or topology awareness can shorten the lookup latency, especially when p_s is around 0.7 for link heterogeneity or around 0.3 for topology awareness. It reveals that the hybrid system is more sensitive to the extensions.

TABLE 2
TOTAL $connum$ UNDER DIFFERENT p_s VALUES.

p_s	TTL=1	TTL=2	TTL=4
0	4882354	4882354	4882354
0.1	4384861	4384861	4384861
0.2	3955554	3955556	3955560
0.3	3455618	3455663	3455683
0.4	3014036	3014192	3014231
0.5	2454076	2454480	2454513
0.6	1903119	1903916	1904185
0.7	1393979	1395594	1396464
0.8	919519	924644	929429
0.9	462057	479646	502686

Table 2 shows the total number of peers the data lookups contact under different TTL values. Generally speaking, structured peer-to-peer networks contact more peers to find a data item than unstructured peer-to-peer networks. This does not mean that unstructured peer-to-peer networks are more efficient than structured peer-to-peer networks because the lookup failure ratios of unstructured peer-to-peer networks are much greater than that of structured peer-to-peer networks. $connum$ decreases linearly with the increase of p_s. When $p_s = 0.9$, we can see that $connum$ is around 10% of that of the structured peer-to-peer networks. This is also due to the three steps of a data lookup. When p_s is small, most of peers are t-peers. A data lookup will pass a half of the t-network on the average. When p_s is large, the number of t-peers involved in the second step decreases. Although the number of s-peers involved in the first and third steps increases, the increase is very slow as the s-peers are evenly distributed in different s-networks.

We can also observe that the TTL value has no impact on $connum$ when p_s is small. However, when p_s is large, the larger the TTL value is, the larger $connum$ is. This is because that a larger TTL value enlarges the range of the flooding packet, thus increases the number of peers involved. Thus we have the following conclusion: the average number of peers a lookup may contact drops linearly as p_s increases. Also, increasing TTL causes slightly greater $connum$ only when p_s is larger than 0.5.

VI. Conclusions and Future Work

In this paper, we have proposed a hybrid peer-to-peer system which combines both the structured peer-to-peer network and the unstructured peer-to-peer networks to form a two-tier hierarchy to provide efficient and flexible distributed data sharing service. The top tier is the t-network which is a structured ring based peer-to-peer network providing efficient and accurate service. The bottom tier is composed of multiple unstructured s-networks which provide approximated best-effort service to accommodate flexibility. By assigning peers to the t-network or the s-network, the hybrid peer-to-peer system can utilize both the efficiency of the structured peer-to-peer network and the flexibility of the unstructured peer-to-peer network and achieve a good balance between them. Our simulation results show that compared to structured peer-to-peer networks, the hybrid system has less lookup latency and $connum$, thus has higher data lookup efficiency. The efficiency can be further increased when considering link heterogeneity and topology awareness. Compared to unstructured peer-to-peer networks, the hybrid system has much lower data lookup failure ratio. By adjusting the system parameter p_s and TTL value, the data lookup failure ratio can be lowered to an acceptable value.

Our future work includes how to design a caching scheme for the hybrid peer-to-peer system to improve the system performance. In the case that some extremely popular data is requested by a large amount of peers, the peer hosting the data may be overwhelmed by the large amount of requests. The goal of the caching scheme is to balance the load of the hosting peer when popular data is requested by many peers. The idea is to distribute the load among as many peers as possible so that no peer is overwhelmed. The challenge is how to choose some surrogate peers to redirect the requests to, which data should be cached and how long the data should be cached.

References

[1] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, “A survey and comparison of peer-to-peer overlay network schemes,” *IEEE Communications Survey and Tutorial*, March 2004.

[2] S. Sen and J. Wang, “Analyzing peer-to-peer traffic across large networks,” *Internet Measurement Workshop*, November 2002.

[3] Y. Chu, S. Rao and H. Zhang, “A case for end system multicast,” *ACM Sigmetrics ’00*, pp. 1-12, Santa Clara, CA, June 2000.

[4] E. Brosh and Y. Shavitt, “Approximation and heuristic algorithms for minimum delay application-layer multicast trees,” *IEEE INFOCOM ’04*, Hong Kong, March 2004.

[5] M. Freedman and R. Morris, “Tarzan: A peer-to-peer anonymizing network layer,” *The 9th ACM Conference on Computer and Communications Security*, November 2002.

[6] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee and S. Khuller, “Construction of an efficient overlay multicast infrastructure for real-time applications,” *IEEE INFOCOM ’03*, pp. 1521-1531, San Francisco, CA, March 2003.

[7] S. Iyer, A. Rowstron and P. Druschel, “Squirrel: A decentralized, peer-to-peer web cache,” *The 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[8] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan and H. Zhang, “The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols,” *IPTPS ’05*, pp. 115-126, Ithaca, NY, February 2005.

[9] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, “Peer-to-peer computing,” *Technical Report HPL-2002-57*, HP Labs, March 2002.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, “A scalable content addressable network,” *ACM SIGCOMM ’01*, pp. 161-172, 2001.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 17-32, 2003.

[12] V. Vishnumurthy and P. Francis, “On heterogeneous overlay construction and random node selection in unstructured P2P networks,” *IEEE INFOCOM ’06*, 2006.

[13] Gnutella development forum, the gnutella v0.6 protocol, http://groups.yahoo.com/group/the gdf/files/

[14] Bittorrent, http://www.bittorrent.com/

[15] P. Ganesan, Q. Sun and H. Garcia-Molina, “YAPPERS: A peer-to-peer lookup service over arbitrary topology,” *IEEE INFOCOM ’03*, pp. 1250-1260, 2003.

[16] S. Ratnasamy, M. Handley, R. M. Karp and S. Shenker, “Topologically-aware overlay construction and server selection,” *IEEE INFOCOM 2002*, New York, NY, June, 2002.

[17] H. Zhang, G. Neglia, D. Towsley, G. Lo Presti, “On unstructured file sharing networks,” *IEEE INFOCOM ’07*, pp. 2189-2197, 2007.

[18] M. Zaharia and S. Keshav, “Gossip-based search selection in hybrid peer-to-peer networks,” *Proc. IPTPS*, February 2006.

[19] http://www.cc.gatech.edu/projects/gtitm/.