Peer-to-Peer Netw. Appl. (2011) 4:3–22
DOI 10.1007/s12083-010-0089-8

PSON: A scalable P2P file sharing system with efficient complex query support

Yan Li · Jyoti Ahuja · Li Lao · Jun-Hong Cui · Shigang Chen

Received: 30 December 2009 / Accepted: 23 July 2010 / Published online: 14 September 2010

Abstract A desired P2P file sharing system is expected to achieve the following design goals: scalability, routing efficiency and complex query support. In this paper, we propose a powerful P2P file sharing system, PSON, which can satisfy all the three desired properties. PSON is essentially a semantic overlay network of logical nodes. Each logical node represents a cluster of peers that are close to each other. A powerful peer is selected in each cluster to support query routing on the overlay network while the less powerful peers are responsible for the maintenance of shared contents. To facilitate query routing, super peers are organized in form of a balanced binary search tree. By exploiting the concept of semantics, PSON can support complex queries in a scalable and efficient way. In this paper, we present the basic system design such as the semantic overlay construction, query routing and system dynamics. A load balancing scheme is proposed to further enhance the system performance. By simulation experiments, we show that PSON is scalable, efficient and is able to support complex queries.

Keywords P2P file sharing · Complex query support · Semantic overlays · Load balancing

Y. Li · J.-H. Cui (✉)
Computer Science & Engineering,
University of Connecticut, Storrs, CT 06269, USA
e-mail: jcui@engr.uconn.edu

Y. Li
e-mail: li2yan@yahoo.com

J. Ahuja
Yahoo Software Development India PVt Ltd,
Bangalore, 560001, India
e-mail: jyoti.ahuja@gmail.com

L. Lao
Google Santa Monica, 604 Arizona Avenue,
Santa Monica, CA 90401, USA
e-mail: llao@google.com

S. Chen
Department of CISE, University of Florida,
Gainesville, FL 32611, USA
e-mail: sgchen@cise.ufl.edu

1 Introduction

Peer-to-peer file sharing has been an active research area since the first massive P2P service for music file sharing (http://napster.com/) was created in 1999. A successful P2P file sharing system should achieve the following design goals. First, the system should scale to a large number of peers spreading throughout wide areas across different administrative domains. Secondly, efficient and effective file lookup must be provided to the users. Location of a requested file should be determined with the minimum communication and computation overhead. Lastly, a desired P2P file sharing system should support complex (or partial-match) queries. As opposite to keyword searching, a complex query is composed by a set of file attributes. For example, a music file can be described by the attributes of singer, composer, title, year, etc. A desired search mechanism should be able to handle queries that contain a subset of full attributes or queries that even have typos.

In the literature, many P2P files sharing systems have been proposed and captured people's attention

[1, 3, 7, 8, 13, 18, 21, 24, 28]. However, very few systems can achieve all the design goals stated above. In this paper, we tackled the issues of scalability, efficiency and complex query support by a new P2P file sharing system, **P**eer-to-peer **S**emantic **O**verlay **N**etwork (PSON). The basic design of PSON effectively exploits the concepts of hierarchy and semantics.

Files shared in PSON are classified based on a prior semantic hierarchy. Figure 1 shows a simple example in which each node represents a semantic entry and all the nodes together form a semantic tree. A semantic entry is identified by its level in the hierarchy and its label such that the semantic entries can be easily compared and ordered. Every file shared in the system semantically belongs to a tree node. Thus, we can sort all the files in the system based on their semantic entries.

In PSON, peers are organized into clusters. Within each cluster, a "super peer" which is more powerful in terms of bandwidth, availability, etc. is selected among all the peers in the cluster, while the rest of peers are "normal peers". Each cluster is assigned a semantic entry and maintains the content directory for all the files shared in the system that belong to the assigned semantic entry. If we consider each cluster as a logical node, all the logical nodes together form a semantic overlay network in which queries are routed on the basis of semantic entries. When a peer publishes a file, it first extracts the meta data for the file and generates a location-meta data pair (i.e., a content directory item). Then the directory item is injected into the semantic overlay network, routed to the responsible "host" cluster, and is finally stored at a normal peer in the cluster. Similarly, if a peer wants to search for a file, it issues a search query based on the semantic entry of the file. The query is then routed to the cluster responsible for storing the content directory of the semantic entry. By a local search in the cluster, the location of the requested item can be obtained.

Clearly, semantic overlay construction and query routing are the fundamental issues in the whole system design. To conserve the semantics of the shared contents and facilitate efficient query routing, we propose to use red-black tree as the overlay structure of the logical nodes. With the red-black tree structure, an in-order tree traversal could yield an encoded semantic hierarchy, which can naturally help to conserve the semantics of the overlay tree. Due to the balanced tree structure, the average query search delay in terms of the number of logical hops on the semantic overlay is bounded by the logarithm of the overlay size.

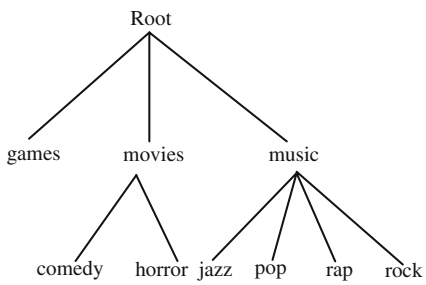


Fig. 1 An example of semantic hierarchy

Another critical issue addressed in this paper is system dynamics, which is a common feature of P2P systems. Peers can join and leave the system at will and files are injected into and revoked from the system at any moment. System dynamics can affect the semantic overlay significantly and must be handled appropriately to guarantee the system's performance. In this paper, we propose some mechanisms to handle peer and file dynamics.

To evaluate the performance of PSON, we conduct extensive simulations using the NS-2 simulator (http://www.isi.edu/nsnam/ns/). Based on the simulation results, we can claim that PSON is a powerful system that can guarantee all the desired design goals.

Besides the basic system design, we also address an open issue due to the tree structure of the semantic overlay: load balancing. We propose to add cross links between peers to distribute queries more evenly in the system. We discuss in detail how cross links are formed and maintained, and evaluate the performance of PSON with the cross link mechanism. With cross links, the load at high level nodes in the semantic overlay can be significantly reduced and the search performance of the system can be improved.

The rest of the paper is organized as follows. In Section 2, we review the background of P2P file sharing and some existing systems. Then in Section 3, we present the PSON architecture and discuss the basic design issues, including overlay construction, overlay routing, and system dynamics. In Section 4, we evaluate the performance of the basic system using simulations. In Section 5, we discuss the issue of load balancing and propose the cross link mechanism. At the end of this paper, we conclude our current work and discuss our future research directions in Section 6.

2 Background and related work

In this section, we first provide some background on the concepts of semantic hierarchy and hierarchical P2P

systems, which are the basis of our work. Then we review some related work in P2P system design.

2.1 Background

2.1.1 Semantic hierarchy

Content shared in a file system can be semantically classified into a hierarchy (referred to as "semantic hierarchy" or "classification hierarchy") based on genres. For example, music files can be first categorized into jazz, pop, rap, and rock, etc, and rock music files can be further subdivided into "soft rock" and "hard rock". Each subcategory of music files can further creates a sub-hierarchy based on composer and year etc. This type of content classification is proved to be very useful in content organization, especially in centralized databases. However, in P2P systems, semantic hierarchy has not been widely explored yet. Crespo et al. [6] pioneered the research in this direction. In [6], nodes with semantically similar contents are clustered together and form an overlay network of clusters. For each semantic group in the hierarchy, an overlay network is created. Therefore, a node can join multiple overlay networks to which it has semantically related contents. In this system, semantic overlay networks are not structured. Thus, *flooding* or a *centralized directory* is used to locate a node that belongs to those overlay networks, which results in very high query search overhead.

2.1.2 Hierarchical P2P system

PSON exploits the heterogeneity of peers, which is similar to KaZaa (http://kazaa.com/), an enormously successful P2P file sharing service. In KaZaa, peers are classified into normal peers and super peers. Super peers usually have more connection bandwidth and better availability. Normal peers connect to a super peer and together form a cluster. Each super peer keeps track of the IP address and the shared content of each normal peer in its cluster. Therefore, a search query is first resolved by the super peer and then forwarded to the normal peers in the cluster that have the requested contents. KaZaa is more scalable compared to the others such as Napster (http://napster.com/) and Gnutella (http://gnutella.wego.com/) due to its hierarchial architecture. However, it cannot support complex queries as query routing is irrespective of semantics. Moreover, the super peers are not structured and thus *flooding* or *partial pooling* is used for the communication between super peers, which is obviously inefficient.

2.2 Related work

2.2.1 DHT based systems

DHT based P2P systems (such as CAN [15], Chord [22], Pastry [17], and Tapestry [29], to name a few) provide efficient data retrieval for exact-match queries. The key idea behind these systems is to assign particular content or pointer to the content to the peers in the system.

A hash function is introduced to map the requested contents to a unique identifier based on a given keyword and the range of the hash function is distributed among all the peers in the network.

DHT based P2P systems are only designed for exact queries since hash functions do not take the semantics of a query into consideration.

2.2.2 P2P systems supporting semantics

Recently, there are several proposals [4–6, 9, 23] which support semantics in P2P systems. In pSearch [23], semantics of a document are generated by applying Latent Semantic Indexing (LSI) on a term (or semantic) vector which is generated from the document using Vector Space Model (VSM). CAN is then employed to create an overlay by using the document semantics as the key to store document index. In principle, this work extends the classical IR (Information Retrieval) algorithms to the P2P environment so as to provide "content" based search. In other words, the "semantics" in pSearch are basically abstracted from the document content. SSW [9] employs a similar approach to the one used in pSearch to generate semantic vectors. In SSW, peers are clustered according to the semantics of local data and self-organized as a small world overlay network. Further, a dynamic dimension reduction method is used to decrease the dimensionality of this overlay network.

The key difference between PSON and these systems is that PSON assumes a prior semantic hierarchy. To some extent, PSON is complementary to these systems. An analogy is that Yahoo classification is in fact an indispensable service to us even though we have powerful Google search. This is because that it is quite often that we do not know the right keywords, but we do know what category our needs belong to.

2.2.3 Hierarchical directory service systems

From the requirement of a prior semantic hierarchy, PSON is quite similar to some hierarchical directory service systems such as DNS [11, 12] and TerraDir

[19, 20]. DNS is one of the most important services widely deployed in the Internet. It resolves domain names through lookup and search in a hierarchical distributed database. TerraDir generalized DNS and designed generic directory services for arbitrary applications. Moreover, advanced replication and caching techniques are employed to improve performance. Compared with these two systems, the key difference is on overlay construction and routing: PSON maps the semantic hierarchy to a red-black overlay tree, while TerraDir and DNS map the namespace directly to the physical network topology. The major advantage of the use of red-black tree is that the routing performance can be significantly improved, and the average search time will be strictly bounded by $O(\log(n))$.

3 PSON architecture

In this section, we first provide a high-level overview of the PSON system design in which we highlight the principles for constructing the semantic overlay. Then we show the detailed algorithms for the key operations: semantic overlay construction and search query routing. We also discuss the critical issue of system dynamics and propose our solutions for peer and file dynamics.

3.1 Overview

PSON is a self-organized semantic overlay network, which consists of a number of logical nodes. A logical node stands for a cluster of peers that are geographically close to each other. Within each cluster, a powerful peer in terms of high bandwidth, better availability, etc. is selected as a super peer that represents its cluster. Different from KaZaa, the super peers are not necessarily maintain the local directories; instead, they are mainly responsible for query routing in the overlay network, i.e. communicating with other super peers as well as coordinating local normal peers. The overlay network constructed by the super peers is well organized based on semantics. Before we present the construction of the semantic overlay, the concept of semantic hierarchy needs to be clarified.

3.1.1 Semantic tree

Files shared in PSON are classified into a semantic hierarchy. In Fig. 1, a tree structure of the semantic hierarchy is illustrated.[1] Each tree node stands for a semantic entry that is denoted as a tuple with three attributes: *semantic label*, *semantic level* and *ancestor list*. As shown in Fig. 1, "Root", "music", "movie", "jazz", "rock", etc. are all semantic labels which describe a class of files. Semantic level is the tree level of the semantic entry in the semantic tree. Ancestor list of a semantic entry is the list of the semantic labels and levels of all its ancestors. For instance, the three attributes of "pop" would be:

Semantic Label: Pop
Semantic Level: 2
Ancestor List: Music 1, Root 0

3.1.2 Ordering semantic entries

With the three attributes, semantic entries can be easily compared and ordered. For any two semantic entries, if they have the same parents, their order is decided lexicographically. If they have different parents, we first find their highest uncommon ancestors (according to their ancestor lists) and compare them lexicographically. For example, referring to Fig. 1, "games" is smaller than "movies" since they have the same parent "Root" and "g" is smaller than "m". Similarly, we can say "comedy" is smaller than "pop" because their highest uncommon ancestors are "movies" and "music" respectively, and "movies" is smaller than "music".

3.1.3 Semantic overlay construction

The key operation in PSON is to construct an overlay network based on the semantic hierarchy. In other words, PSON needs to organize various semantic entries into a well-structured overlay. Once the semantic tree is determined, the simplest way for overlay construction is to do a direct mapping from the semantic tree to an overlay network. For example, we can easily construct an overlay tree which has exactly the same shape as the semantic tree. However, content search in such an overlay network is not guaranteed to be efficient as the semantic tree may have undesired shape, eg. very unbalanced. Moreover, we have to keep in mind that P2P systems usually present high dynamics: files are frequently inserted and deleted, and peers can join and leave the system at any moment. To maintain a dynamic P2P system in an efficient and

[1] Semantic tree is probably the most popular and useful classification hierarchy. Other hierarchies are also possible as shown in [6]. In our work, we use the tree structure to illustrate the design of PSON.

scalable way is very challenging. Thus, selecting a good overlay structure and mapping the semantic tree to the overlay are the key issues in the system design.

Overlay structure In our design of PSON, we use balanced binary search tree (i.e., red-black tree) as the overlay structure. We select red-black tree because of its many desired features: (1) an in-order tree traversal can yield an ordered semantic list; (2) Search time is bounded by logarithm of the tree size; (3) The tree construction and maintenance are simple. It is possible to employ other overlay structures, such as mesh, ring, etc. based on different needs. For system enhancement, we actually convert the red-black tree into a mesh as shown in Section 5.

From semantic tree to overlay tree In PSON, the red-black overlay tree is built in an incremental fashion. In the overlay tree, each cluster (or logical node) corresponds to one semantic entry in the semantic tree. In other words, the peers in the cluster maintain the directory of the files belonging to the semantic class. However, the "mapping" cluster of a semantic entry only "appears" in the overlay network when necessary. For example, at early stage of the system, only a few files are shared among peers, so it is not necessary to build a big overlay tree. In this case, it is very likely that only one cluster is enough to maintain all the content directories, and thus the overlay tree is actually a "degraded" tree with only one node (which corresponds to the "Root" semantic entry). Later on, with more music files being shared, we "fork" a cluster that is dedicated to maintain music file directories and the overlay grows to a two-node tree. Then how to connect the new logical node to the existing overlay? Since we use binary search tree as the overlay structure, the new node is inserted into the existing tree based on semantics following the insertion algorithm of red-black tree. For instance, "music" is smaller than "Root", then the node storing "music" files will be on the "left" side of the node storing "Root" files (or unclassified files). In such a manner, we can construct a semantic overlay in the form of red-black tree.

In the following, we discuss in detail the two basic PSON design issues: overlay construction and overlay routing.

3.2 Building a semantic overlay

As mentioned earlier, the overlay network is built in an incremental fashion. Clusters/logical nodes are inserted in such a way that they form a balanced binary tree structure. The insertion procedure has two steps. First, a logical node is inserted into the existing tree according to its semantic entry; Secondly, an insertion fixup procedure is invoked in order to preserve the red-black tree properties. The insertion fixup method may include some rotation operations to make the tree balanced.

Every node in the overlay tree keeps pointers to its left child, right child, parent, the smallest descendant in its left subtree, and the largest descendant in its right subtree. The pointers to the smallest and largest descendants are kept for query routing purposes. Algorithm 1 shows the logical node insertion procedure. In this algorithm, the pointers maintained by a logical node are denoted by "node.left", "node.right", "node.parent", "node.smallest", and "node.largest". "root" is the root of the overlay tree, while "Root" is the first semantic entry in the semantic hierarchy.

From the Algorithm 1, we can see that in the semantic overlay, the node of "Root" is pre-existing in the system with level 0. Any new node is created only by its semantic parent. This design is very intuitive: if there are only a few files in a subcategory, it is not necessary to construct a new overlay node to store the corresponding directory information and the node for the parent category should be sufficient. Next we

Algorithm 1 Overlay-insertion(*newnode*)

```
1: // "⟨", and "⟩" denotes semantic comparison
2: Contact bootstrap node, B
3: Add newnode's address to the super peer list in B
4: B checks if newnode is the first super peer
5: if true then
6:    root=newnode
7:    newnode.right = Root
8:    newnode.largest = Root
9:    Root.parent = newnode
10: else
11:    Strategically select one existing super peer, and
       return address to newnode
12:    newnode contacts this super peer (referred to as
       oldnode)
13:    if newnode ⟨ oldnode then
14:       Recursively find the position of newnode in the
          left subtree of oldnode and insert it into the
          tree
15:    else
16:       Recursively find the position of newnode in the
          right subtree of oldnode and insert it into the
          tree
17:    end if
18: end if
```

explain the overlay construction algorithm with the help of an example depicted in Fig. 2.

Initially, "Root" is the only node in the system and handles all types of files. Then we assume that the number of "music" files increases beyond a certain upper bound and a new node dedicated for "music" is requested. "Root" initiates the insertion operation for the new node that has the semantic label "music" and level 1. The directory of music files is moved to this new node which can further create its own semantic children if necessary. Let "rap" be the next node to be inserted. The insertion procedure of "rap" is initiated by its semantic parent "music". The location of "rap" in the overlay is determined as follows: "music" first checks whether or not "rap" lies in its own left subtree by comparing "rap" semantically with its smallest descendant. Since "music" does not have a left descendant in the current overlay, it checks to see if "rap" belongs to its parent's left subtree or not. "music" does not have a parent either because it is currently the root of the overlay tree; thus, it makes "rap" its own left child. "movie" and "pop" are then inserted by their respective semantic parents, "Root" and "music" in similar procedures. Clearly, the overlay tree is not balanced after node insertions and we need to make it balanced in order to support efficient query routing.

A right rotation is performed by "rap" to restore the balanced structure of the overlay tree. After the rotation, the subtree rooted at "rap" gets one level closer to the root and the subtree at "music" becomes one level further from the root of the overlay tree. As shown in Fig. 2, the overlay tree is balanced afterwards. In our algorithm, a rotation is a local operation and is bounded by $O(1)$ time. A rotation is invoked whenever the overlay tree becomes unbalanced due to node insertion/deletion. Since each node in a red-black tree maintains a "color" bit, no global information (for example, the hight of the (sub)tree) is needed to decide if a rotation should be performed or not. Thus, rotations in the semantic overlay tree are performed in a decentralized way.

Maintaining the semantics In Fig. 2, the overlay nodes are inserted in the following order : "Root", "music", "rap", "movie", "pop". Figure 3 shows the resultant overlay after all the semantic entries in Fig. 1 are inserted. The in-order walk on the tree is: "games", "comedy", "horror", "movie", "jazz", "pop", "rap", "rock", "music", "Root". Though the overlay construction depends on the order in which individual nodes are inserted, it is guaranteed that for the same semantic hierarchy, an in-order traversal on its mapped overlay tree yields the same ordering of the semantic entries irrespective of the insertion order. Thus, the insertion algorithm can maintain the semantics.

Overlay node deletion The cost for overlay node deletion is comparatively less than the cost for insertion. The reason is that for insertion, the location of the new node in the overlay must be searched first before a fixup is performed. While in the case of deletion, the node can directly leave the overlay without any extra operations (at the maximum, it tells its semantic parent that it is leaving the overlay). After each deletion, a fixup procedure is invoked to restore the red-black tree properties of the overlay which may again involve new rotations.

3.3 Overlay query routing

If a normal peer issues a search query, the query will first be processed by its local cluster. If its local cluster does not have the requested files, the peer will pass the query to its super peer, which then initiates an overlay query routing procedure. The overlay query routing algorithm is shown in Algorithm 2.

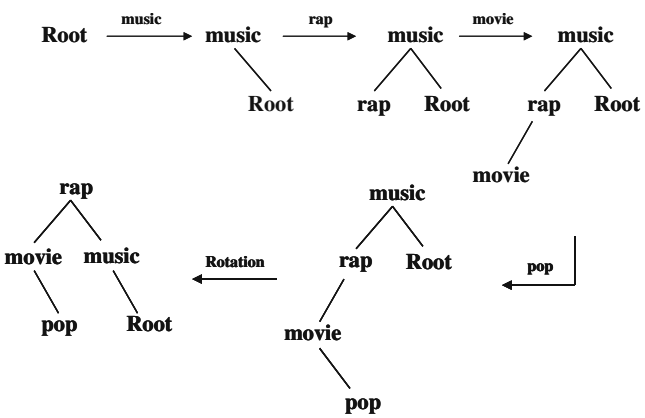


Fig. 2 An example of overlay node insertion

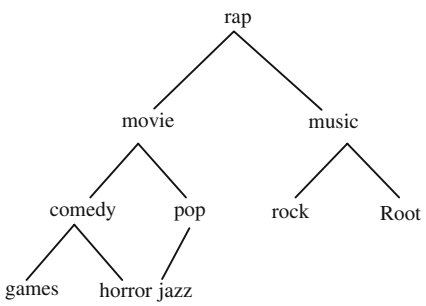


Fig. 3 The structure of the overlay tree

Algorithm 2 Overlay_Query(*current_node*,*query_node*)

```
1: // ⟨,⟩ denotes semantic comparison
2: if current_node ⟩ query_node then
3:    if query_node ⟨ current_node.smallest || current_node.smallest == null then
4:       pass the query to current_node.parent
5:    else
6:       pass the query to current_node.left
7:    end if
8: else
9:    if query_node ⟩ current_node.largest || current_node.largest == null then
10:      pass the query to current_node.parent
11:   else
12:      pass the query to current_node.right
13:   end if
14: else
15:   the query is resolved
16: end if
```

An example is illustrated in Fig. 4. Consider a query for "rock" songs from a cluster that is responsible for the directory of "comedy" movies. "comedy" is compared with "rock" (Line 2) and since "rock" is greater than "comedy", it is now compared with the largest descendant of its right subtree at "comedy" (Line 9) i.e. "horror". "rock" is even greater than "horror", therefore, it is further forwarded to the parent of "comedy", "movie" (Line 10). "movie" performs the same comparison again (Line 2) and routes the query accordingly.

3.4 Complexity analysis

From the above description, we can easily conclude that the complexity of both overlay insertion/deletion and query routing is $O(\log(n))$ due to the fact that red-black tree is utilized as the overlay structure (where n is the number of overlay nodes). If there are totally N peers and the average cluster size is M, the average number of operations for a single insertion/deletion and query search is bounded by $O(\log(N/M))$. If we compare with the average search time of SSW,[2] which is $O(\frac{\log^2(2N/M)}{l})$ (where l is a constant and is set to 4 in the evaluation of SSW [9]), we can claim that query search in PSON is much faster than SSW in average.

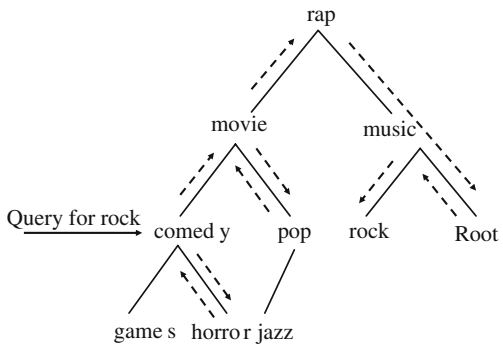


Fig. 4 An example of query routing. *Dashed arrows* denote the query path

3.5 How system dynamics affect the overlay?

A common feature of a P2P file sharing system is high dynamics: peers can join and leave the system at any moment and the files can be inserted into and deleted from the system frequently. System dynamics can affect the overlay significantly. In the following, we propose our strategies of handling peer and file dynamics in PSON.

3.5.1 Peer dynamics

Peer join Insertion of a new peer into the system might lead to a *new cluster formation*. When a peer joins the system, it first constructs connections with a few existing peers which are geographically close to it [16, 25]. Then the new peer will try to cluster with its neighbors based on node connectivity in the sense that peers are highly connected in the same clusters and sparsely connected between clusters. The clustering procedure should follow a distributed clustering algorithm [10, 14]. A new cluster will be created in the following scenarios: (1) When there is no candidate cluster for the new peer to join; and (2) when the number of normal peers in the desired cluster exceeds its maximum limit. We propose to keep a maximum size limit for a cluster to achieve better performance and minimize the maintenance overhead. More specifically, the maximum size limit is measured by cluster diameter. We want to maintain tightly connected clusters such that peers in the same clusters are geographically close to each other and the cost for local search within each cluster can be minimized. A cluster is not assigned a directory to manage until the number of peers in that cluster exceeds a minimum threshold. This is necessary because there would be a lot of overhead for storing a directory due to excessive dynamics if the number of

[2]We have identified the fundamental difference between SSW and PSON in Section 2. Here, we simply compare the average search path length of the two systems.

peers in a cluster is too small. This strategy ensures that the existing cluster handles all requests while the new cluster is being created. This is significant as the system can provide continuous service during configuration changes.

Peer leave Events of peer leave from the same cluster can lead to the cluster decay. If the number of peers in a cluster decreases beyond a certain threshold, the directory assigned to that cluster is revoked and may be moved to another cluster, because there will be a lot of overhead for a small number of peers to handle a considerably sized directory. Further, if there are two such clusters physically close by in the network, we will combine them and a new directory is assigned to the merged cluster. On the other hand, if there is no others available for merging, the cluster remains without a directory to manage and waits for new member peers.

3.5.2 File dynamics

File insertion Addition of files to a directory may lead to its split. We propose to keep the directory size within a maximum limit so that if the current size increases beyond that limit, the directory is split into two and the other half is assigned to another cluster that has no directory to manage. This is performed in order to improve the search efficiency and keep low overhead for directory management.

File deletion Successive file deletion from a single cluster may lead to significant decrease in the size of the managed directory. In this scenario, we plan to (1) "merge" two such directories so that member peers of that cluster have enough content to manage or (2) merge the shorter directory with its semantic parent. It also helps to achieve file load balancing among clusters.

In the next section, we evaluate the performance of PSON for each operation proposed above using simulations, focusing on overlay construction and query search.

4 Simulation study

In this section, we evaluate the performance of PSON using NS-2 simulations. We focus on the performance of the basic PSON system in overlay construction and query search. Then, we evaluate how system dynamics can affect the performance.

4.1 Simulation setup

We implement PSON in NS-2, and conduct simulation experiments on Redhat Linux platform. The performance of the system is evaluated at two levels as described in the following.

4.1.1 Simulation of overlay construction and query search

For overlay construction and query routing, we measure the system performance at the overlay level, i.e. we do not consider the local search within a cluster. The simulation is initialized by having one logical node (with semantic label as "Root") pre-exist in the semantic overlay and then inserting logical nodes into the existing overlay tree till the system reaches a certain size. Afterwards, we simulate the events of logical node leave and query routing.

The performance of the basic PSON system at the overlay level is measured by the following metrics:

- **Search Delay** is the delay for locating a requested (class of) file(s). We measure this metric by averaging the number of logical hops traversed by a query from the requesting node to the destination.
- **Search Success Ratio** is the percentage of search queries for which the destination can be successfully located.
- **Insertion Delay** is the delay for inserting a new logical node at its proper location in the semantic overlay tree. This metric is measured as the average number of logical hops traversed by an insertion request in the overlay. The overhead of rotation is not considered in this metric.
- **Stabilization Overhead** is the delay for the overlay network to stabilize itself after an insertion or deletion. After an insertion, the semantic overlay needs a fixup procedure to maintain its balanced structure that may involve some rotations. Deletion of a logical node itself does not cause any delay but the overlay tree may still need a fixup procedure to restore its balance. This metric is measured by the total number of logical hops traversed by an insertion or deletion message including the extra hops for fixup and rotations.

4.1.2 Simulation of system dynamics

The performance of peer and file dynamics is evaluated at the level of peers: we consider the local search within each cluster as well as on the overlay. The metric is

Search Delay that includes the number of logical hops within a cluster.

4.2 Performance of the basic system

In this set of simulations, we measure the performance of the basic PSON system. The physical network topologies are generated using the transit-stub model from the GT-ITM topology generator [27]. Following are some important issues related to the experiments.

- **Semantic Tree Generation** A semantic hierarchy is the input to the simulation. To create a semantic hierarchy, we generate a set of random words with 5 letters. Each random word stands for a class of files (such as "Music", "Comedy") that a logical node is responsible for, and characterizes the label of a semantic entry. Each word has a random number of children. The height of the semantic hierarchy is also under control. In our simulations, we set the total number of semantic entries as 1000, and the maximum height of the semantic hierarchy as 20. In addition, the number of semantic children of each node is under control. For all the simulations, we set the maximum number of semantic children as 3 unless clarified otherwise.
- **Query Generation** The semantic hierarchy created in the above is maintained at each node to facilitate query generation. To closely mimic query behaviors in real world, we generate overlay queries in the following way: Whenever a node is prompted to issue a query, it randomly selects a semantic entry from the predefined semantic hierarchy, and then releases the selected semantic entry as the query into the overlay network.
- **Query Classification** We design two types of queries: exact queries and complex queries. An exact query with semantic entry α requires the system to return the meta data for the files which exactly match α back to the requesting node. While for a complex query with semantic entry β, the system should return the files which match β as well as the files match any of the "descendant" semantic entries of β. For instance, if "music" and two of its semantic children, "jazz" and "classical" are present in the overlay, a complex query for "music" requires the location of "music", "jazz" and "classical". This is because a complex query for "music" most probably means that *I do not know what music files to download, but I want to browse all music files*. Thus "jazz" and "classical" which are semantic descendants of "music" should also be located. Since each query can be mapped to a node in the semantic hierarchy, an exact query is considered successful if the overlay node responsible for the same semantic entry is found. For a complex query, it is termed as successful if all of the overlay nodes classified in the semantic subtree of the query are found. For instance, "music", "pop" and "rap" are the only music nodes in the overlay. If all the three nodes are returned for the query "music", we will claim that the query is 100% successful as it finds all the nodes that can be classified as "music". It should be noted that PSON can indirectly handle arbitrary queries, such as "join", "union", "select", and "projection". These types of queries should be first decomposed into multiple queries (including exact and complex queries) which can be routed directly in PSON. In other words, a relatively powerful user interface at each peer is required to decompose arbitrary queries, collect search results and conduct post-query processing. In reality, such a user interface is not difficult to implement. On the other hand, how to incorporate in-network query processing in PSON is an interesting topic in our future research plan.

4.2.1 Insertion/deletion cost

We measure the delay for logical node insertion/deletion and the cost to stabilize the overlay after logical node dynamics. For simplicity, the node insertion and deletion events are not interleaved.

Node insertion In this experiment, we measure the delay that it takes for the overlay to add a new logical node at its proper position. Figure 5 shows the number

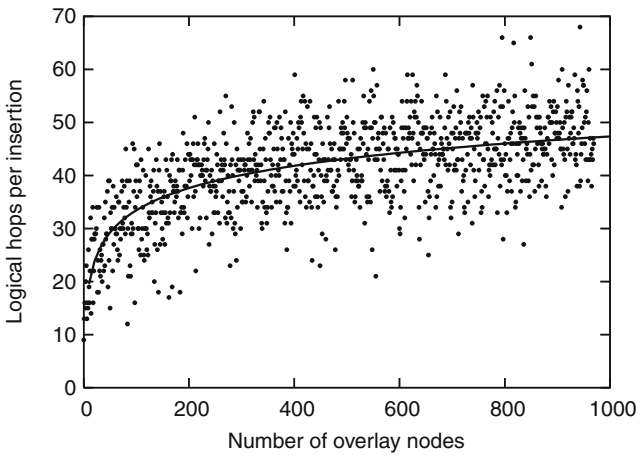


Fig. 5 Effect of overlay size on insertion performance

of logical hops traversed per overlay insertion. The total number of logical nodes is kept constant at 1,000. At the beginning, there is only one logical node "Root". Then the overlay nodes are inserted into the system one by one, and thus the size of the overlay tree kept increasing. Note that the results for this experiment does not include the logical hops traversed in rotations that are carried out to balance the overlay tree after inserting a new node. Clearly, the average number of logical hops for an overlay node insertion is in logarithm of the overlay size, which is consistent with the balanced binary tree structure. The logarithmic nature indicates the overlay construction is scalable to a large overlay size.

Insertion fixup A new node insertion may cause unbalanced structure of the semantic overlay tree. Therefore, it is important to incorporate an insertion fixup mechanism after each insertion event. The fixup mechanism checks if any of the essential properties of a red-black tree is violated and will restore the balanced tree structure if necessary. Some rotations might be performed in the fixup procedure, and hence some extra overhead in terms of the number of logical hops may be caused for an overlay node insertion. Figure 6 shows the total number of logical hops traversed by an insertion message including insertion fixup. We can see that as we increase the size of the overlay, the stabilization overhead also increases for some of the insertions. This is because an insertion fixup operation can cause more fixups if the tree is large. An insertion fixup actually works on a part of the tree, and when carried out on that part it may un-balance other parts and trigger another fixup which involves more rotations. This process propagates until the whole overlay tree is balanced. Moreover, there is a relatively larger variation in the insertion overhead with the fixup mechanism implemented compared with the overhead without fixup. This is because not every node insertion triggers the fixup mechanism. Only if a new node insertion breaks the balanced binary tree structure, a fixup procedure is invoked. Also, not every fixup triggers a chain of further fixups. Therefore, a node insertion may generate different amount of overhead depending on whether or not fixup procedure is needed.

Node deletion An overlay node deletion does not cause any delay on its own because no search is involved to locate the node to be deleted and the node can just simply leave the system. However, a fixup procedure must be carried out after each deletion event to check and restore the red-black tree property. Rotations may be triggered by the fixup mechanism to balance the overlay. To capture the overhead caused by node deletion, we measure the number of logical hops traversed by a fixup after a deletion is performed.

Figure 7 depicts the overhead for different number of deletion events. For each deletion, a node is deleted right away and the deletion fixup method is invoked. It is evident from the results that the number of hops traversed by a deletion message decreases with the increased number of deletions because of the reduced overlay size. Also, we can see that some deletion events generate more overhead than the others especially when the overlay size is larger. The higher overhead is mainly caused by more rotations during the fixup procedure. For some nodes in the overlay, their deletion can cause unbalanced tree structure, for which rotations must be carried out and more overhead is generated.

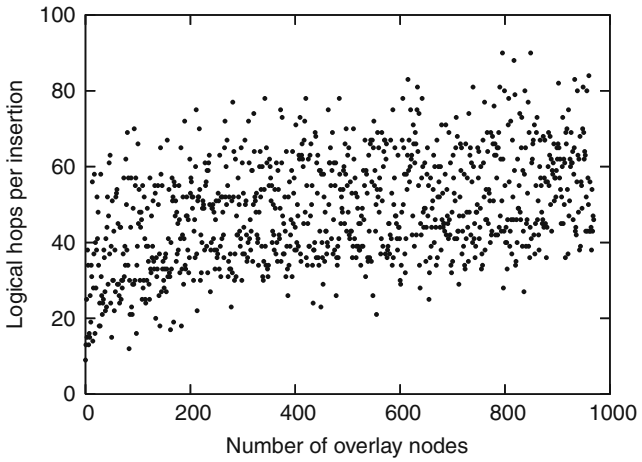


Fig. 6 Number of logical hops traversed per insertion including fixup

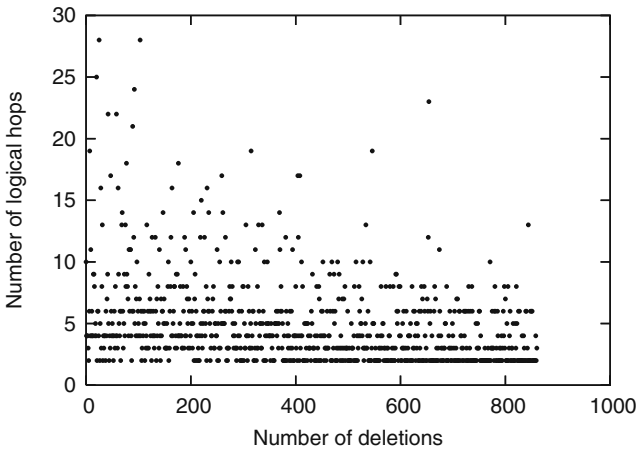


Fig. 7 Average number of logical hops traversed per deletion fixup

Moreover, some fixup procedures may trigger further fixups in a chain, which results in comparatively higher overhead.

4.2.2 Query performance

In this experiment, we evaluate the performance of the basic PSON system on query search. The average search delay for both exact and complex queries are measured.

Exact query performance First we present the performance of PSON on handling exact queries. If the logical node being searched is present in the system, the search mechanism guarantees that it will be found in $O(\log(n))$ hops where n is the number of logical nodes. The lower curve in Fig. 8 shows the average number of logical hops traversed (i.e., the search delay) for exact queries. We can observe that the query performance is nicely bounded by the logarithm of the overlay size and hence is very efficient in large systems.

Complex query performance Now we show the performance of PSON when handling complex queries. The delay experienced for complex queries is also shown in Fig. 8. Compared with the exact queries, the complex queries traverse a larger number of logical hops on average. This is consistent with our assertion that more logical nodes will be returned as the destinations as a complex query does not supply enough semantic attributes. However, the average search delay of complex queries is again bounded by $O(\log(n))$. Therefore, we can claim that PSON is able to support complex queries efficiently.

4.2.3 Comparison of PSON and flooding overlay

In this experiment, we compare PSON with flooding overlay (used in SON [6], which is the most related work to PSON in semantic overlay design) for the search performance on both exact and complex queries.

In flooding overlay, a search query received by a node is forwarded to all of the neighbors. Every neighbor then checks whether it can answer the query. If a match is found, it replies to the originator of the query; otherwise it forwards the query to its neighbors and the same operations are repeated. In this way, a search query must be processed by every node in the overlay network, which is clearly not scalable. One way to curtail the search process is using TTL-controlled flooding mechanism. Each query carries a TTL field. Whenever a node receives a query, it decrements the TTL value. If the TTL becomes zero, the forwarding process is terminated. TTL value effectively reduces the query propagation in the system, but it also lowers the chance to successfully locate destinations. Therefore, the performance of flooding overlay is sensitive to the value of initial TTL.

For exact queries, we evaluate flooding overlay against three different TTL values, 5, 10 and 20. Figure 9 shows the search delay and search success ratio of PSON and flooding overlay under different number of overlay nodes. The percentage values correspond to the search success ratio. In flooding overlay, the success ratio decreases with the number of overlay nodes for each specific TTL value (except for TTL = 20). This is because the percentage of nodes receiving a query message decreases with the increased overlay size. PSON, on the other hand, achieves 100% search success ratio

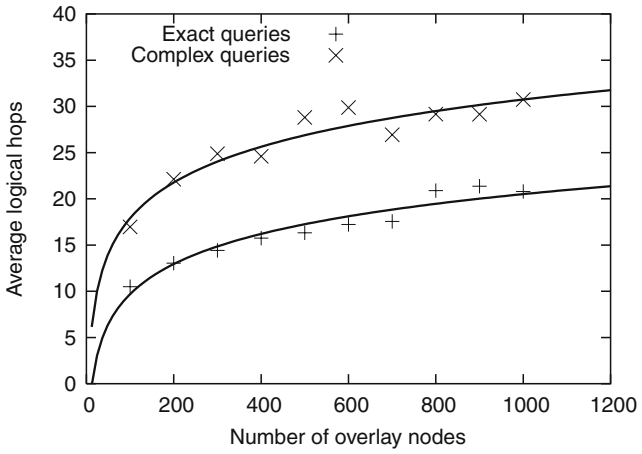


Fig. 8 Search delay with varying overlay size

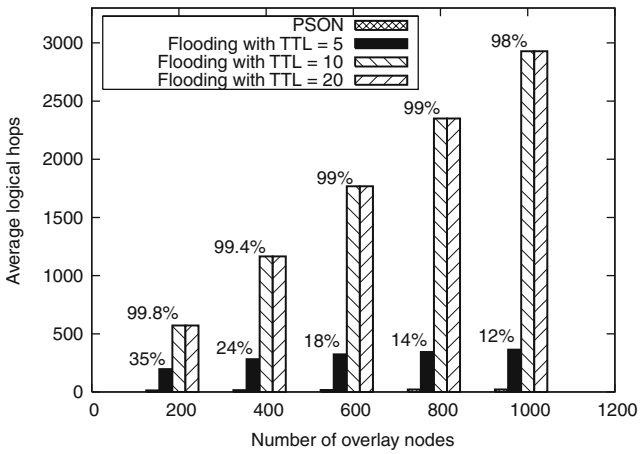


Fig. 9 Comparing exact query performance of PSON and flooding overlay

on all the overlays. In addition, even though the search success ratio of flooding overlay with TTL 20 is also 100%, PSON achieves the same performance with a significantly lower search delay. According to our experiment results, the search delay of PSON is within five percent of the average logical hops traversed in flooding overlay to successfully locate a search query.

For the complex query experiments, we obtain similar results, which are omitted from this paper due to space limit.

4.2.4 Effect of semantic hierarchy on PSON's complex query performance

Another factor that has significant impact on the performance of PSON is the distribution of non-leaf nodes in the semantic hierarchy. The average search delay for a complex query in PSON is related to the distribution of leaf queries released into the system. Since the queries are selected randomly from the semantic hierarchy, the performance of PSON is dependent on the distribution of leaf semantic entries. If the number of leaf nodes is large, the average search delay is small because more search queries are for leaf nodes and will immediately stop once they reach the logical nodes responsible for the leaf queries. However, if a non-leaf query is found, the system needs to proceed to find all the semantic descendants of that non-leaf node. It is worth mentioning that leaf and non-leaf queries have the same effects on search delay if they are not found: the system will search for the first semantic ancestor of the query that is present in the overlay. This implies that the distribution of leaf nodes in the semantic hierarchy can alter the query search performance of PSON.

Figure 10 illustrates the average search delay of PSON as we vary k, the maximum number of child nodes of a semantic entry in the hierarchy. As shown in the results, the average number of logical hops for a search query increases with the size of the overlay. The simulation results corroborate our expectation that the change in a semantic hierarchy alters the performance of PSON. If we decrease the maximum number of semantic children, the number of non-leaf nodes as well as the percentage of non-leaf queries increases. As a result, the average number of logical hops traversed for a search query increases for the overlays with the same size.

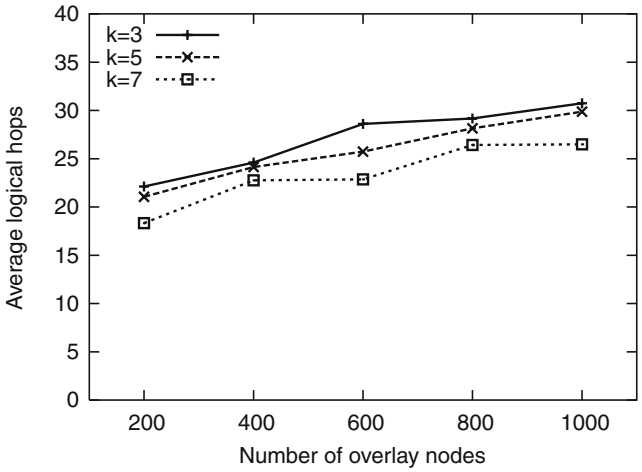


Fig. 10 Effect of number of overlay nodes on average number of logical hops for 1,000 exact queries

4.3 System dynamics

In this set of simulations, we focus on the performance of PSON in handling system dynamics. We consider two different types of network topologies: random topologies generated by Waxman [26] and power law topologies by BA [2]. We want to evaluate whether high performance can be kept even when the system is highly dynamic, for which the main metric is search delay which includes the number of logical hops traversed within a cluster.

4.3.1 Peer dynamics

We consider two types of events in the system: concurrent peer join and leave. When a new peer joins the system, it first tries to join a cluster of its neighbors. The requirement for clustering is that the cluster diameter should not exceed 3. If a new peer can not find an existing cluster to join, a new cluster is created for this peer. The cluster formation is handled by the distributed clustering algorithm SDC [10] which has good performance in forming clusters based on node connectivity. When an existing peer leaves the system, it needs to transfer the meta data it maintains to the other members in the cluster. Through simulations, we study the effects of simultaneous node join and leave on the system's performance in query search.

We consider a topology with 1,000 nodes and clustered by SDC. 1,999 files with semantic entries randomly selected from the predefined semantic tree are injected into the system so that an overlay network is created to maintain the meta data of these files. In our simulation, each peer can maintain the meta data for up to ten files. For node join events, X peers are inserted into the system simultaneously and the search delay is measured after the join events. Similarly, to simulate node leave, X existing peers are randomly selected and removed from the system. If a cluster becomes

too small to maintain the file information, it will be replaced by the largest spare cluster. We control X from 0 to 200 in order to simulate different level of dynamics.

Figures 11 and 12 show the search delay of PSON in dynamic systems. Even when the system is highly dynamic with 200 simultaneous Join/Leave, the average number of hops traversed by a search query is still quite close to the value measured from the static system. This observation verifies that our mechanism in handling peer dynamics can successfully keep the high performance of the system.

4.3.2 File dynamics

In our simulation of file dynamics, we focus on the performance of PSON in handling file insertion. After successive file insertion, the number of file information maintained by a cluster might exceed the capacity of the cluster given each peer can maintain at most ten files. In this situation, we randomly pick a child node of the semantic entry that the current cluster is responsible for based on the predefined semantic tree and move all the meta data that belongs to the child entry to a spare cluster. Then the spare cluster is assigned the child semantic entry and is inserted into the semantic overlay network.

We should be aware that the effect of file dynamics is highly related to cluster formation. If a cluster has a larger size, it is capable of maintaining more files and will be affected less by file insertion. However, large clusters may not guarantee good search performance: if the cluster has large diameter, local search within the cluster can become very expensive. Therefore, in order to achieve good search performance under file dynamics, a clustering method that can form tightly connected clusters with small diameters is necessary. The existing clustering algorithm SDC is used for this purpose.

We consider the two types of topologies, random and power law, with 1,000–5,000 nodes. To simulate file dynamics, we insert 1,999–9,999 files into the system and measure the search delay which includes local search within a cluster. Figure 13 shows the performance of PSON in handling file dynamics on the two types of topologies. With the increase of topology size, the search delay increases slowly and steadily. This is because when more files are inserted into the system, more clusters will be inserted into the semantic overlay. The growth of the overlay increases the number of hops traversed for a search query. However, with large increase of file insertion, the search delay is not affected dramatically.

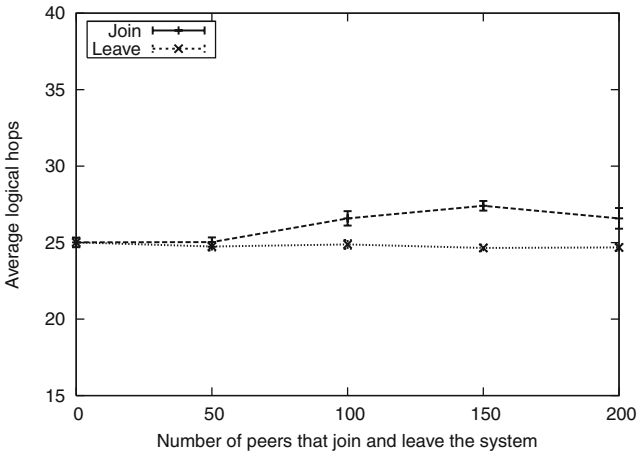


Fig. 12 Search delay in dynamic system with power law topology and 1,000 peers

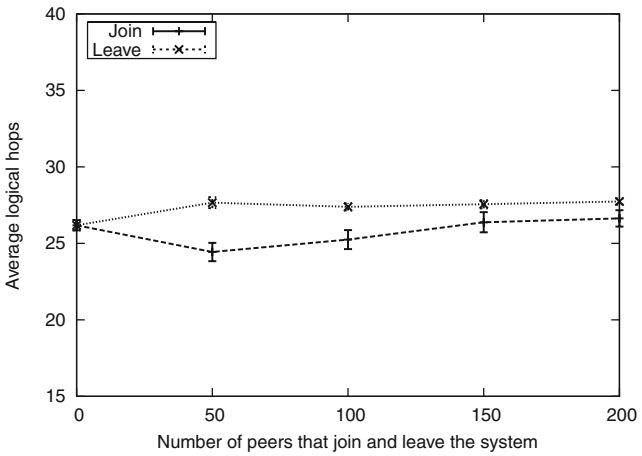


Fig. 11 Search delay in dynamic system with random topology and 1,000 peers

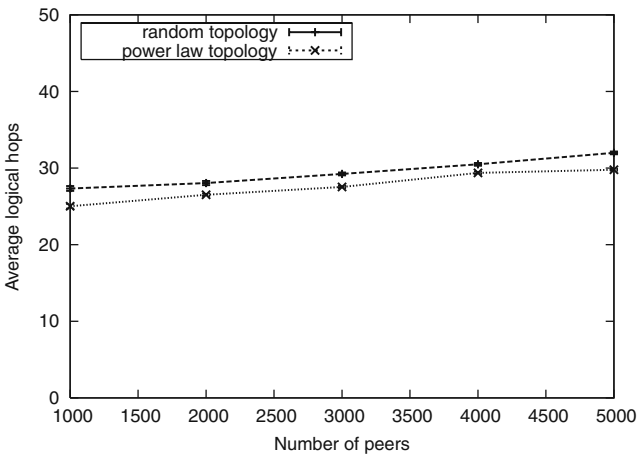


Fig. 13 Search delay under file dynamics with 1,000–5,000 peers

4.4 Summary

In this section, we have presented the performance of PSON in simulation environment. Overlay insertion/deletion operations are shown to be scalable. The lookup efficiency for both exact match and complex queries is bounded in logarithm of the overlay size. In comparison with flooding overlay, PSON achieves significantly better query performance in terms of the number of logical hops and search success ratio. We also learn from the simulations that the structure of the predefined semantic hierarchy have impacts on PSON's query performance.

Besides overlay construction and query search, we also study the effects of system dynamics on the performance of PSON. Based on the simulation results, we can claim that PSON is able to keep good search performance even with high peer and file dynamics.

5 System enhancement

As shown above, PSON presents many desirable features. However, we admit that the basic system design does not provide an effective solution for query traffic load balancing. Specifically, the balanced binary tree structure of the system can result in high query traffic at the root and high level nodes in the semantic overlay, assuming queries are distributed uniformly among all the peers in the system. To address this issue, we propose a novel and effective load balancing mechanism to further enhance the system: we add cross links between logical nodes based on the query traffic flow.

A cross link is defined as a unidirectional virtual link between a pair of overlay nodes (represented by their super peers). The origin of a cross link needs to maintain the IP address and the related semantic information of the end. Next, we discuss the cross link mechanism in detail.

5.1 Construction of cross links

We denote the node which initiates a query routing as 'S' and the node at which the routing path ends as 'D'. Here a node is referred to the super peer of a logical node in the overlay. Since queries are routed on the basis of semantics, we can always locate the destination 'D'. To create a cross link, we select the source node 'S' as the origin and determine the end of the cross link by a uniform random sampling on the nodes along the routing path.

Suppose the length of the routing path is k. Then the first node n_1 on the path is sampled with probability $\frac{1}{k}$. If n_1 is not selected based on the sampling probability, the next node on the routing path, n_2 will be sampled with a probability $\frac{1}{k-1}$. Without loss of generality, the ith node along the routing path is sampled with a probability $\frac{1}{k-i+1}$. It is easy to prove that the sampling probability defined in such a way can guarantee all the nodes along the routing path are selected with equal probability. Once an intermediate node is sampled as the end of a cross link, it will send its own IP address and its semantic information back to 'S' and the subsequent nodes on the routing path will not be sampled any more.

However, a real implementation issue needs to be addressed here: we can not obtain the length k of a routing path before the routing gets started. An estimation on k is needed by the sampling scheme. The method we use to estimate the routing path length k is similar to the well-known RTT estimation. We define a new field l in the query packet to measure the path length of a specific routing query. At node 'S', let k_{i-1} be the estimated path length before the ith query Q_i is issued. l is initialized as 0 and is incremented by 1 at each intermediate node traversed on the path. Thus, at the destination 'D', the value of l is exactly the length of the routing path for the query Q_i. Then node 'D' will simply send l back to the source node 'S'. 'S' will then estimate the routing path length after i queries as: $k_i = (1-\alpha)l + \alpha k_{i-1}$, where α is a constant less than 1. Then in the $i+1$th query, 'S' can sample the end of a new cross link using the estimated query path length k_i.

One issue encountered in this construction scheme is the additional overhead caused by returning the value of l at the end of the query routing. We solve this problem by returning the value of l with some probability. For instance, node 'D' will return l back to 'S' with a probability p.

5.2 Maintenance of cross links

Each node needs to refresh its own cross links periodically to remove the stale ones due to the high dynamics of the system. The cost for maintenance will be huge if the node has a large number of cross links. It is necessary to bound the total number of cross links at each node. We will show in our simulation that an upper bound as small as 10 can balance the load distribution effectively.

5.3 Routing with cross links

We assume that queries are uniformly distributed in the system, i.e. half of the total queries will be routed from one side of the semantic overlay to the other side. In

the absence of cross links, nodes at the higher levels of the overlay will receive and process a higher percentage of the total query traffic in the system. With cross links, we expect some of the traffic will be routed directly to the other side of the tree/sub-tree without passing to the high level nodes.

In our new routing algorithm, before a query is forwarded to the next hop, the current node will check if the destination lies in the sub-tree of the end of a maintained cross link by comparing the query with the semantic entry of each maintained cross link end. Among all the the cross link ends that have the destination in the sub-tree, one node will be randomly selected as the next hop of the query. Then the routing is continued at the selected cross link end. If none of the cross link ends has the destination in their subtree, the current node holding the query will simply forward it based on the original routing algorithm.

To illustrate the operations, let's consider a query for "rock" at node "comedy" as shown in Fig. 14. We assume the node "comedy" has two cross links to "music" and "jazz" respectively. "comedy" first compares "rock" with itself. Semantically, "rock" is larger than "comedy". Under the new routing algorithm, the query "rock" will not be forwarded to the largest descendant of "comedy" at this point. Instead, "comedy" compares the query with its two cross link ends " music" and "jazz" and decides to forward "rock" to "music" directly as the query "rock" must be in the subtree of "music". After "music" receives the query, it performs a semantic comparison with itself and "music" is larger than "rock". Since "music" doesn't have any cross links, it follows the original routing algorithm and forwards the query to the destination. This way, the high level nodes which will be traversed based on the original routing algorithm, such as "movie" and "rap" are not involved in the new routing path at all. Thus, their load is reduced.

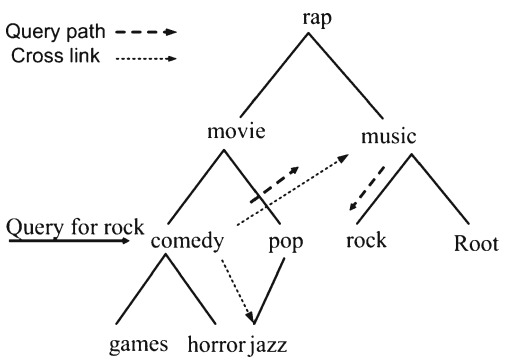


Fig. 14 Query routing with cross links

Since under the new routing algorithm a query is forwarded to a cross link end only when it is in the subtree of that node, the search time will be surly bounded by $O(\log(n))$.

5.4 Performance with cross link mechanism

In this set of experiments, we examine the effect of the load balancing scheme on the performance of PSON. Some important settings related to the simulations need to be clarified as follows.

- **Threshold for the number of cross links** As mentioned in previous section, a threshold on the maximum number of cross links maintained at each node needs to be defined to limit the maintenance overhead. In our simulations, we fix this threshold as 10.
- **Probability p of returning query path length** This probability affects the estimation on the query path length used in the cross link construction. For all the simulations, p is set as a constant value: 0.4.

We study the performance of the load balancing scheme under two different network states: **transient state** in which the average number of cross links is still in the growing stage and **steady state** in which the average number of cross links gets stable, i.e. close to the threshold 10. The transient state is considered because it better reflects the cross link construction overhead as well as the influence of the number of cross links on the performance of the system. For the steady state, we focus on the performance of cross links in different overlay network size.

5.4.1 Simulation results in transient state

This set of simulations are conducted on a semantic overlay of 300 nodes. The number of cross links at each node is determined by the amount of queries issued: it increases with the number of queries until gets to the predefined threshold. As shown in Fig. 15, for 2,000 queries, the average number of cross links maintained in the system is less than 4. It grows linearly and gets converged after 6,000 queries are issued.

The first metric we are interested in is the average traffic load in the system (Fig. 16). It is observed that the average traffic load increases steadily with the number of queries. By adding cross links in the system, we can reduce the traffic load significantly. For example, the average traffic load for 6,000 queries is reduced from 400 to 300 by having nine cross links at each node on average. Moreover, the advantage of the cross link mechanism is more significant if more cross

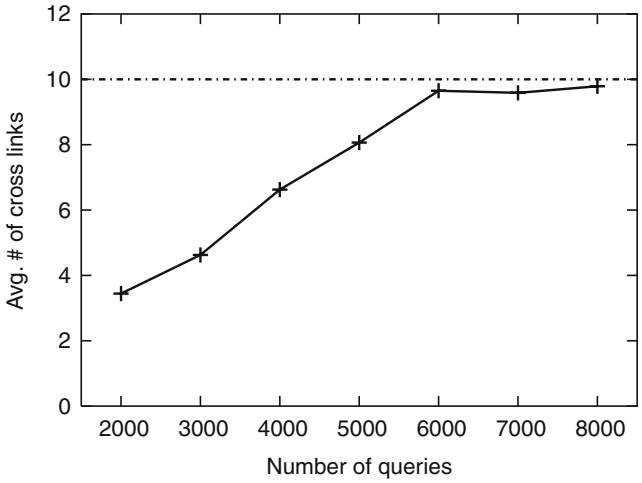


Fig. 15 Average number of cross links with the increase of search queries

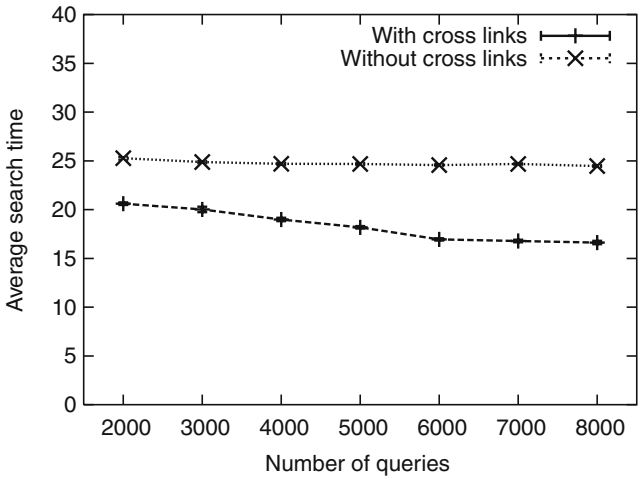


Fig. 17 Average search time with increased number of queries

links are constructed, as shown by the increased gap between the data points in the figure. It indicates that the construction and maintenance overhead for a cross link is trivial compared with the benefit brought by a new cross link.

Figure 17 shows the average search time under different number of queries. For the system without cross links, since the overlay structure is not changed once generated, the average search time is quite stable. When cross links are added to the system, the search time can be decreased a lot. Also, the decreased amount is enlarged by more cross links as more queries can be routed directly to the other side of the system without traversing the high level nodes.

5.4.2 Simulation results in steady state

In this set of simulations, we study the performance of the load balancing scheme after the number of cross links gets stable, i.e. close to the predefined threshold. Our focus is on the performance of the cross link scheme for different overlay network size.

We first study the traffic load at the root of the overlay tree which can be over loaded in the original PSON. Figure 18 shows that with cross links implemented, the traffic load at the overlay root is decreased significantly: reduced to approximately $\frac{1}{3}$ of the amount without cross links, which indicates the cross link scheme is able to balance the traffic load at root effectively.

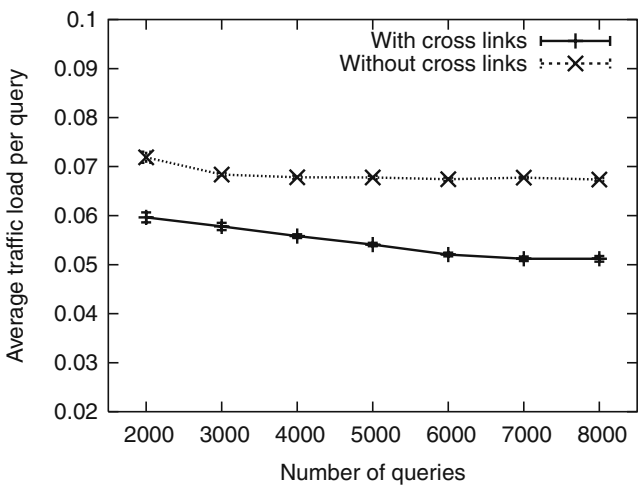


Fig. 16 Average traffic load with increased number of queries

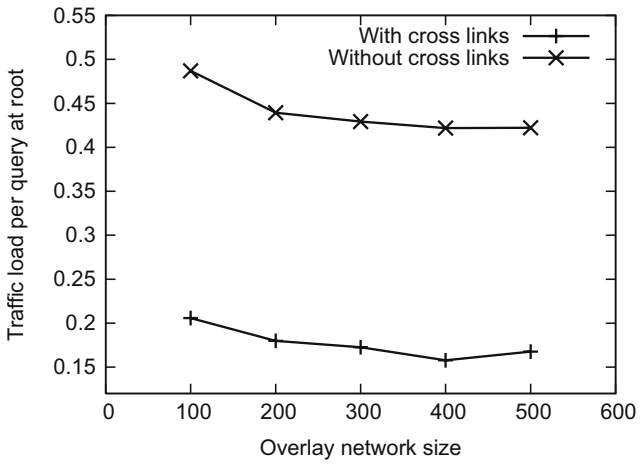


Fig. 18 Traffic load distribution at overlay root with 100–500 overlay nodes

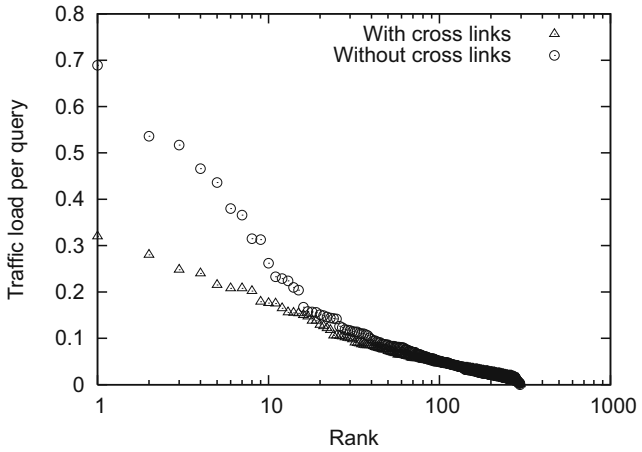


Fig. 19 Traffic load distribution in the overlay with 300 node

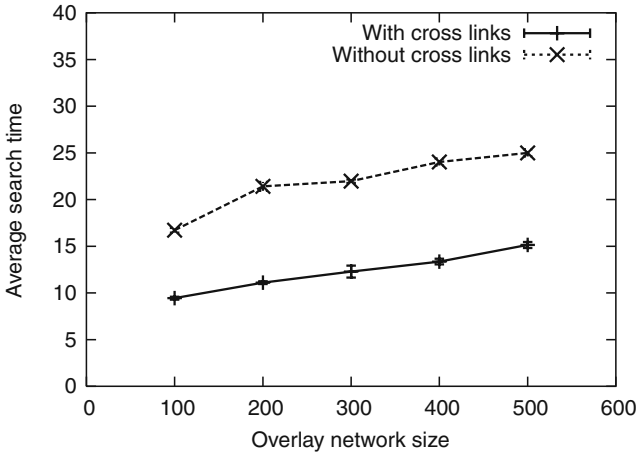


Fig. 21 Average search time with 100–500 overlay nodes

We also measured and compared the distribution of traffic load after cross links are implemented with the original system. For an overlay with 300 nodes as shown in Fig. 19, the traffic load is much more evenly distributed in the system with cross links constructed.

The above simulation results demonstrate our cross link mechanism can successfully achieve the goal of load balancing in PSON.

Figure 20 shows the average traffic load for different overlay network size. Clearly, with more overlay nodes joining the system, queries are more distributed. Thus the average traffic load has a decreasing trend. With the cross link mechanism, the average traffic load is further decreased, which indicates an improved routing efficiency. Consistently, the average search time of a query in the system is also reduced with the implementation of cross links as shown in Fig. 21.

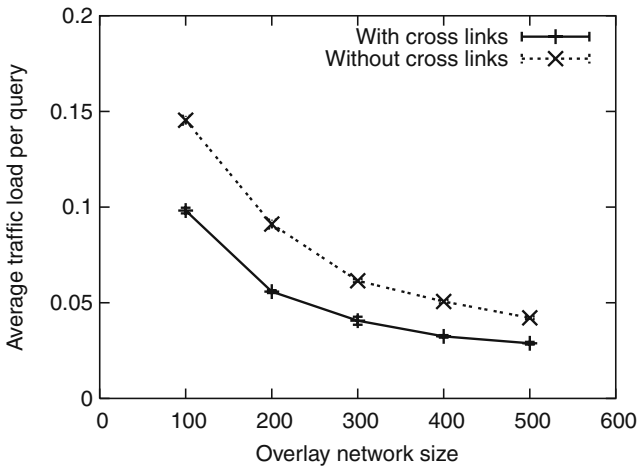


Fig. 20 Average traffic load with 100–500 overlay nodes

6 Conclusions and future work

In this paper, we propose a powerful P2P file sharing system, PSON. The design of PSON effectively exploit the concept of semantics and hierarchy. We present the PSON architecture and the key issues of overlay construction and search query routing. By simulation experiments, we claim that PSON can satisfy all the design goals of a successful P2P file sharing system.

- Scalability: PSON is not only scalable to a large number of peers, it is also scalable to a large amount of traffic.
- Routing efficiency: By constructing a semantic overlay with a structure of red-black tree, PSON is able to achieve logarithmic search delay, which is highly efficient for large systems.
- Complex query support: By exploiting the concept of semantics, PSON is able to support complex queries very effectively.

Besides the design of the basic system, we also propose a new load balancing scheme to further enhance the PSON system. By adding cross links along the query routing flows, the traffic received by the root and the high level overlay nodes can be effectively reduced, and hence the traffic load in the system can be effectively balanced.

To make PSON a powerful system in various network scenarios, there are still several issues to explore. (1) PSON does not consider geography-aware overlay routing, as obviously might lead to long search delay even though the number of logical hops is bounded. Designing an overlay routing algorithm which could incorporate the physical topology is definitely desirable. (2) Construction of the semantic overlay depends on

a predefined semantic hierarchy. It is likely that the semantic hierarchy may grow or shrink with time. How to adapt the existing semantic overlay for the change of the semantic hierarchy is an interesting topic to study. (3) A smart method to determine the maximum number of cross links maintained by each logical node is desired. We want to balance between the overhead for maintaining the cross links and the more evenly distributed traffic load as well as the more efficient query routing due to the cross link mechanism.

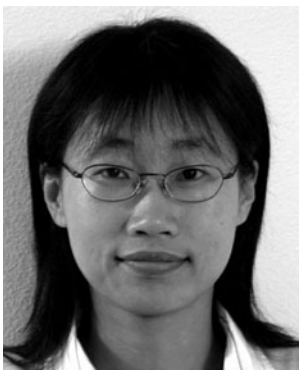
References

1. Aberer K, Cudré-Mauroux P, Hauswirth M, Pelt TV (2004) Gridvine: building internet-scale semantic overlay networks. In: International semantic web conference, pp 107–121
2. Barabasi A-L, Albert R (1999) Emergence of scaling in random networks. Science 286:509–512
3. Bharambe AR, Agrawal M, Seshan S (2004) Mercury: supporting scalable multi-attribute range queries. ACM SIGCOMM Comput Commun Rev 34(4):353–366
4. Chen Y, Xu Z, Zhai C (2005) A scalable semantic indexing framework for peer-to-peer information retrieval. In: SIGIR 2005 workshop: heterogeneous and distributed information retrieval
5. Cohen E, Fiat A, Kaplan H (2003) Associative search in peer-to-peer networks: harnessing latent semantics. IEEE INFOCOM 2:1261–1271
6. Crespo A, Garcia-Molina H (2004) Semantic overlay networks for P2P systems. In: International workshop on agents and Peer-to-Peer computing (AP2PC'04), pp 1–13
7. Doulkeridis C, Norvag K, Vazirgiannis M (2007) Desent: decentralized and distributed semantic overlay generation in P2P networks. IEEE J Sel Areas Commun 25(1):25–34
8. Huang Y, Fu TZ, Chiu D-M, Lui JC, Huang C (2008) Challenges, design and analysis of a large-scale P2P-VOD system. SIGCOMM Comput Commun Rev 38(4):375–388
9. Li M, Lee W-C, Sivasubramaniam A (2004) Semantic small world: an overlay network for Peer-to-Peer search. In: ICNP, pp 228–238
10. Li Y, Lao L, Cui J-H (2006) SDC: a distributed clustering protocol for Peer-to-Peer networks. In: The fifth IFIP networking conference, vol 3976, pp 1234–1239
11. Mockapetris PV (1987) Domain names—concepts and facilities. Request for Comments 1034. Internet Engineering Task Force
12. Mockapetris PV (1987) Domain names—implementation and specification. Request for Comments 1035. Internet Engineering Task Force
13. Qiu D, Srikant R (2004) Modeling and performance analysis of bittorrent-like Peer-to-Peer networks. SIGCOMM Comput Commun Rev 34(4):367–378
14. Ramaswamy L, Gedik B, Liu L (2005) A distributed approach to node clustering in decentralized Peer-to-Peer networks. IEEE Trans Parallel Distrib Syst 16(9):814–829
15. Ratnasamy S, Francis P, Handley M, Karp R, Schenker S (2001) A scalable content-addressable network. In: ACM SIGCOMM, pp 161–172
16. Ratnasamy S, Handley M, Karp RM, Shenker S (2002) Topologically-aware overlay construction and server selection. In: INFOCOM
17. Rowstron A, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale Peer-to-Peer systems. Lect Notes Comput Sci 2218:329–350
18. Sahin OD, Gulbeden A, Emekci F, Agrawal D, Abbadi AE (2005) Prism: indexing multi-dimensional data in P2P networks using reference vectors. In: MULTIMEDIA '05: proceedings of the 13th annual ACM international conference on multimedia. ACM, New York, pp 946–955
19. Silaghi B, Bhattacharjee B, Keleher P (2002) Query routing in the terradir distributed directory. In: SPIE ITCOM, vol 4868, pp 299–309
20. Silaghi B, Gopalakrishnan V, Bhattacharjee B, Keleher P (2004) Hierarchical routing with soft-state replicas in terradir. In: The 18th international parallel and distributed processing symposium
21. Sripanidkulcha K, Maggs B, Zhang H (2003) Efficient content location using interest-based locality in Peer-to-Peer systems. IEEE INFOCOM 3:2166–2176
22. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM, pp 149–160
23. Tang C, Xu Z, Dwarkadas S (2003) Peer-to-Peer information retrieval using self-organizing semantic overlay networks. In: ACM SIGCOMM, pp 175–186
24. Terpstra WW, Kangasharju J, Leng C, Buchmann AP (2007) Bubblestorm: resilient, probabilistic, and exhaustive Peer-to-Peer search. SIGCOMM Comput Commun Rev 37(4):49–60
25. Waldvogel M, Rinaldi R (2003) Efficient topology-aware overlay network. SIGCOMM Comput Commun Rev 33(1):101–106
26. Waxman BM (1988) Routing of multipoint connections. IEEE J Sel Areas Commun 6:1617–1622
27. Zegura EW, Calvert KL, Bhattacharjee S (1996) How to model an internetwork. IEEE INFOCOM 2:594–602
28. Zhang R, Hu YC (2005) Assisted Peer-to-Peer search with partial indexing. IEEE INFOCOM 3:1514–1525
29. Zhao BY, Kubiatowicz JD, Joseph AD (2001) Tapestry: an infrastructure for fault-tolerant wide-area location and routing. UC Berkeley, Tech. Rep. UCB/CSD-01-1141

Yan Li received her B.S. degree in Computer Science and Communication Engineering from Southwest Jiaotong University, China in 2002. From 2004 to 2010, she worked as a Ph.D student in the Ubiquitous Networking Research Lab leb by Dr. Jun-Hong Cui at the University of Connecticut. She worked for ECI Telecom from August 2009 and joined Conviva One year later as a software engineer.

Dr. Li's research interests cover the design, modeling, and performance evaluation of networks and distributed systems. Her research mainly focuses on exploiting the spatial properties in modeling of network topology and group membership, scalable and efficient communication support in overlay and peer-to-peer networks.

Jyoti Ahuja received her B.S in Computer Science from Agra University, India and her M.S. degree in Computer Science from University of Connecticut. Currently, She is working for the web search team of Yahoo! Software Development India Pvt. Ltd, Bangalore. Her interests include peer to peer networks, web scale information retrieval techniques and real time stream processing systems. Recently, her work has been focused on user personalization for Yahoo! Search. She has been involved with research on information retrieval areas and has published couple of papers for an internal tech conference.



Li Lao received her B.S. degree from Fudan University, China in 1998. She received her M.S. and Ph.D. degrees in Computer Science from University of California, Los Angeles in 2002 and 2006, respectively. She joined Google Inc in April 2006. Her research focuses on multicasting, overlay network management, multicast modeling and performance evaluation.

Jun-Hong Cui received her B.S. degree in Computer Science from Jilin University, China in 1995, her M.S. degree in Computer Engineering from Chinese Academy of Sciences in 1998, and her Ph.D. degree in Computer Science from UCLA in 2003. Currently, she is on the Faculty of the Computer Science and Engineering Department at University of Connecticut. Her research interests cover the design, modelling, and performance evaluation of networks and distributed systems. Recently, her research mainly focuses on exploiting the spatial properties in the modeling of network topology, network mobility, and group membership, scalable and efficient communication support in overlay and peer-to-peer networks, algorithm and protocol design in underwater sensor networks.

She is actively involved in the community as an organizer, a TPC member, and a reviewer for many conferences and journals. She has served as a guest editor for Elsevier Ad Hoc Networks on two special issues (one on underwater networks and the other on wireless communication in challenged environments). She now serves as an Associate Editor for Elsevier Ad Hoc Networks. She co-founded the first ACM International Workshop on UnderWater Networks (WUWNet'06), and she is now serving as the WUWNet steering committee chair. Jun-Hong received US NSF CAREER Award in 2007 and ONR YIP Award in 2008. She is a member of ACM, ACM SIGCOMM, ACM SIGMOBILE, IEEE, IEEE Computer Society, and IEEE Communications Society. More information about her research can be found at http://www.cse.uconn.edu/∼jcui.

Shigang Chen received his B.S. degree in computer science from University of Science and Technology of China in 1993. He received M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems

for three years before joining University of Florida in 2002. His research interests include network security and wireless networks. He received IEEE Communications Society Best Tutorial Paper Award in 1999 and NSF CAREER Award in 2007. He was a guest editor for ACM/Baltzer Journal of Wireless Networks (WINET) and IEEE Transactions on Vehicle Technologies. He served as a TPC co-chair for IEEE IWQoS 2009, the Computer and Network Security Symposium of IEEE IWCCC 2006, a vice TPC chair for IEEE MASS 2005, a vice general chair for QShine 2005, a TPC co-chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE ICC, IEEE Globecom, etc.