JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1427-1445 (2006)

Interest-Based Lookup Protocols for Mobile Ad Hoc Networks[*]

YI-CHUNG CHEN AND JANG-PING SHEU
Department of Computer Science and Information Engineering
National Central University
Chungli, 320 Taiwan

Peer-to-peer networks and *mobile ad hoc networks* (MANETs) share the same characteristics of self-organization, decentralization, and dynamic topology. Therefore, it is natural to apply peer-to-peer techniques to MANETs. A lookup protocol, one of the most important issues in a peer-to-peer computing, is an essential component for resource searching. In this paper, we propose interest-based bandwidth-efficient lookup protocols, *simple lookup protocol* and *advanced lookup protocol*, for mobile environments. A peer willing to search files broadcasts a query message with keywords relevant to its interests to its neighbors in the transmission range, and only those neighbors also interested in the query forward. Simulation results show that our protocols have higher success rate and raise the scalability and bandwidth efficiency comparing to the previous work. Besides, our protocols can avoid selfish behaviors, since the behavior of forwarding queries benefits not only the source node but also the forwarding node.

Keywords: file sharing, lookup protocol, mobile ad hoc network, peer-to-peer, distributed computing

1. INTRODUCTION

With flourishing development of Internet, more and more resources and information are shared on the Internet. The resources and information may be in the forms of web pages, documents, multimedia files, services, and computing powers, *etc*. Traditionally, resources to be shared are located in a dedicated server and all clients request the server for resources. This is so-called client/server architecture. However, client/server architecture encounters some problems such as fault-tolerance, scalability, load balance, and so on. Peer-to-peer (P2P) networking provides a new paradigm for information exchanges. The emerging of peer-to-peer system is started by Napster, which allows sharing of MP3 music files among Napster's users. Napster provides MP3 file sharing by a centralized directory server which only maintains basic addressability and availability information about the user and the meta-information about the shared files. The user exchanges MP3 files directly from other peers after retrieving the location information.

Peer-to-peer networking is different from the client/server architecture where all the resources are located in a server. Resources in a peer-to-peer network are distributed and peers play both roles of servers and clients simultaneously. Peers sometimes called *servents* (servers/clients) exchange their resources from each other. Because of the charac-

Received October 7, 2004; revised April 12, 2005; accepted June 22, 2005.
Communicated by Chin-Teng Lin.

[*] This work was supported in part by the National Science Council of Taiwan, R.O.C., under grant No. NSC 93-2213-E-008-001

teristics of file sharing, peer-to-peer systems have some attractive properties: the more users engage in, the more resources are shared, and the amount of resources in a peer-to-peer system grows over time. Today, several peer-to-peer systems have been developed such as Napster [17], KaZaA [13], eDonkey [4], eMule [5], and Morpheus [16]. Because of the novelty of peer-to-peer computing, there are still lots of issues to discuss such as social issues, intellectual property rights, lookup protocols, bandwidth efficiency, load balance, streaming, data dissemination, security, reputation, pricing and business model [9, 18, 20, 25].

With the increase of mobile devices as well as progress in wireless communications, ad hoc networking is drawing more and more attention. A mobile ad hoc network (MANET) consists of mobile devices communicating with each other using multi-hop wireless links without any central control. Many potential applications of ad hoc networking have been proposed such as home networking, sensor networks, personal area networks (PANs), and ubiquitous computing. Peer-to-peer networks and MANETs share the same characteristics of self-organization, decentralization, and dynamic topology. Hence, it is natural to apply peer-to-peer techniques to MANETs. Although there have been significant research efforts in peer-to-peer systems during the past few years, peer-to-peer systems for mobile environments is still a new research issue. 7DS [19] is the first approach to apply peer-to-peer technique to mobile environments. It is an architecture and set of protocols supporting Web browsing by on-the-fly file sharing among peers that are not necessarily connected to the Internet. For queries, 7DS implements a multi-hop flooding algorithm combined with multicast delivery of queries. PDI [15] provides the lookup service by locally broadcasting query messages and response messages, and it eliminates the need for flooding the entire network with query messages by maintaining an index cache at every device.

To address the problem of service discovery in a mobile environment, we propose a *simple lookup protocol* (*SLP*) and an *advanced lookup protocol* (*ALP*) for MANET. In contrast to the previous work, our protocols are more bandwidth-efficient and scalable by an interest-based selective forwarding scheme. In SLP, the source node broadcasts a Query message with an interest threshold and a time-to-live (TTL), and only the neighbor nodes whose interests in query string are larger than the threshold keep forwarding the query. The query process goes on until the TTL is equal to zero. All nodes that have the corresponding resources return a QueryHit containing file references reversely along the query path to the source node. In ALP, we save the bandwidth further by determining threshold and TTL dynamically according to feedbacks from previous query requests. Note that, we propose lookup protocols for resources discovery, but do not discuss how the files transmit. The data transmissions after lookup can be solved by using any existed routing protocols such as *dynamic source routing*, *destination-sequenced distance-vector routing*, and *ad hoc on-demand distance vector routing* [11].

The rest of this paper is organized as follows. Section 2 gives an overview of peer-to-peer lookup model and some related works. Section 3 presents our lookup protocols. In section 4, we evaluate the performance of our protocols through simulations. Finally, we conclude our contributions in section 5.

2. PRELIMINARIES

In this section, we introduce lookup models and some related works. A lookup service, one of the most important issues in peer-to-peer computing, is an essential component for resource searching. Several lookup protocols (sometimes called location protocols) have been proposed such as Gnutella [7], Freenet [2], Chord [26], CAN [21], Pastry [23], and PDI [15].

2.1 Peer-to-Peer Lookup Models

Peer-to-peer systems can be classified into centralized, decentralized, and hybrid models [18]. In a centralized peer-to-peer system, the lookup service is provided by a central directory server which maintains basic addressability and availability information about the peers and the meta-information about the resources. In decentralized topologies such as Freenet and Gnutella, all the peers play equal roles. The lookup service is made through propagating query requests. A hybrid (centralized + decentralized) peer-to-peer system is designed for a heterogeneous network and formed by several subnets consisting of one *super-peer* and a limited number of *leaf-peers* having a centralized relationship to the super-peer. In hybrid systems, super-peers bear heavier burdens and run the lookup process with each other in a decentralized manner. Leaf-peers only need to ask their super-peers for responses. Examples of this system are KaZaA and Morpheus.

For decentralized and hybrid peer-to-peer systems, there are two types of lookup protocols currently proposed. One is flooding-based lookup protocol where the peers propagate query requests until the termination conditions meet. This scheme allows more flexible resource placement and search, and it is simple and robust even when frequently joining and leaving of peers, but it is not well scalable because of generating huge messages on querying. Gnutella protocol is a typical flooding-based lookup protocol. The other type of lookup protocols is based on distributed hash table (DHT) such as Chord [26], CAN [21], and Pastry [23]. DHT-based lookup protocols institute strict rules for resource placement and search. Peers are organized into a well-defined *logical* structure, and the answer to a query is in a bounded number of hops. Although DHT-based protocols are more scalable, it also generates huge overhead of maintaining key distribution which is triggered by peers' joining/leaving the network or creation of the keys.

DHT-based protocols are not suitable to be applied in a mobile environment. This is because logical neighbors in DHT-based protocols are probably physically many hops away. In a mobile environment, we take physical topology and physical distance seriously instead of logical ones. Frequently joining and leaving of peers resulting from mobility also challenge applicability of DHT-based protocols. Contrarily, flooding-based protocols are more resistant to the dynamic environment. Therefore, we base our protocols on flooding-based approach.

2.2 Chord

In Chord [26], resource information is a pair of (key, value). A Chord-based application would store and find each value at the node to which the value's key maps. Chord provides just one operation: given a key, it maps the key onto a node. All nodes are organized into a logical identifier circle (or called *Chord ring*), and the protocol uses *con-*

sistent hashing [12] to assign each node and key an identifier and distribute keys evenly among the nodes of the network. Key *k* is assigned to the first node whose identifier is equal to or follows (the identifier of) *k*. This node is called the *successor node* of key *k*, denoted by *successor* (*k*). Queries for a given identifier could be passed around the circle via these successor pointers until they encounter a pair of nodes that straddle the desired identifier; the value in the pair is the node the query maps to. Although Chord is scalable and guarantees to finish a lookup in bound of $O(\log n)$ messages, where n is the number of peers in the network, it is not suitable to be implemented to be a real peer-to-peer system. First, it is very difficult for a node to know who its successor and predecessor are. Second, whenever a node joins or leaves, certain keys must be redistributed to other nodes and finger tables must be updated too. Because of dynamic nature of peer-to-peer networks, arrivals and departures of nodes are occurred quite frequently in a peer-to-peer network, especially in a mobile environment, and maintenance overhead will be quite huge.

2.3 Gnutella

Gnutella [7, 22] is a protocol for distributed search. In this model, participants, called *servents*, play both roles of servers and clients. The Gnutella protocol defines the way in which servents communicate over the network. In order to join the system, a new servent initially connects itself to some of several known hosts that are almost always available. When a servent wishes to search for some files, it broadcasts a query message to its neighbors which forward the query message for its own neighbors. In the dynamic environment, nodes often join and leave and network connections are unreliable. To cope with this environment, after joining the network, a node periodically *pings* its neighbors to discover other participating nodes. Using this information, a disconnected node can always reconnect to the network. Some variation protocols [3, 8] have also been proposed for operating Gnutella on hybrid topologies and (re)configuration algorithms [6] have also been proposed for maintaining the topologies in the MANETs. Opposed to Gnutella, our protocols don’t need to discover and maintain connections. Additionally, we avoid flooding the whole network, and hence reduce the network load.

2.4 Interest-Based Shortcuts

In [25], the authors propose an approach in which peers loosely organize themselves into an interest-based structure on top of the existing Gnutella network. The approach exploits a simple, yet powerful principle called interest-based locality, which posits that if a peer has a particular piece of content that one is interested in, it is very likely that it will have other items that one is interested in as well. When using interest-based shortcuts, a significant amount of flooding can be avoided, making Gnutella a more competitive solution. In addition, shortcuts are modular and can be used to improve the performance of protocols including distributed hash table schemes. Finally, interest-based shortcuts often resolve queries quickly in one peer-to-peer hop, while reducing the total load in the system. Opposed to interest-based shortcuts, our protocols are dedicated to a mobile environment, where frequently joining and leaving of peers occur and the assumption of logical topology of interest-based shortcuts is invalid.

2.5 Passive Distributed Indexing (PDI)

In [15], the authors introduce *Passive Distributed Indexing* (*PDI*), a simple approach for file searching in mobile environments. Each mobile device maintains a *repository* consisting of a set of files stored in the local file system and an *index cache* which stores pairs of keywords and document identifiers. Query-messages and response-messages are transmitted using local broadcast. To overcome the shortcomings of low wireless transmission range of the communication interfaces, messages may be forwarded for a predefined number of hops. All mobile devices listen for broadcasted responses, even if they did not actually issue a query. When a response is received, all reported references to matching documents are added to the local index cache for all keywords contained in the original query. The replacement of index cache entries is conducted in a least-recently-used fashion. Passive distributed indexing implicitly replicates results for popular queries in index caches at several mobile devices. The authors in [15] show that PDI achieves satisfactory success rate and eliminates the need of flooding the whole network with query messages. However, it still causes considerable network overhead due to broadcasting not only query-messages but also response-messages. Opposed to PDI, we use selective forwarding to reduce the number of query messages needed, and the responses are return backwards along the query path to the source node.

3. OUR PROTOCOLS

Traditionally, the lookup service is provided by flooding queries over the whole network. This method is effective to find out matching results but it is not efficient and produces huge network overhead. Especially in the wireless environment, it also causes large number of collisions and leads to degradation of the network performance.

In this section, we present our *simple lookup protocol* (*SLP*) and *advanced lookup protocol* (*ALP*) for MANETs. Our lookup protocols both achieve bandwidth efficiency and high utilization of reference cache via interest-based mechanism. In contrast to the general lookup protocol improving the performance by peers storing related information of query path, which changes frequently in wireless environment, our protocols achieve the bandwidth efficiency by peer storing information of <resource, peer> pairs in the local cache which is resistant to the mobile environment. Efficiency of bandwidth is improved further in ALP by referring to feedbacks from previous query requests. We present the details of two protocols in the following.

3.1 Simple Lookup Protocol

Since traditional flooding mechanism is not bandwidth-efficient, we propose an interest-based lookup protocol selectively propagating queries. Our mechanism operates under the following two assumptions. Firstly, every participant in the peer-to-peer system has its own interests. Secondly, peers will request the resources which he or she is interested in, and the downloaded files reflect its interests. A peer willing to search files broadcasts a query message with keywords relevant to its interests to its neighbors in the transmission range, and only those neighbors also interested in the query forward. If there are any files found, the responses are returned to the peer which issued the query.

To implement SLP, each mobile device maintains a *repository* and a *reference cache*. A repository stores the resources obtained from other peers in the local file system. A reference cache contains the information about the remote resources. An entry in the cache is composed of two fields: file reference File_Ref and access time Access_Time. A File_Ref is a pair of File_Src and File_Info. A File_Src field indicates the location information of a resource such as IP or MAC address of a mobile device. A File_Info field, which could be file name or the meta-descriptions of the file, is used for query matching. The Access_Time indicates the last time the entry is accessed. We define two types of messages, Query and QueryHit, for our protocols. A Query message contains query string *Q* consisting of one or more keywords, *TTL*, and threshold *Thresh* for query matching. *TTL* is the maximum number of hops a message can be forwarded in the network. Threshold *Thresh* is a minimum interest value for query forwarding. A QueryHit message contains file references.

A peer willing to search files broadcasts a Query message with an interest threshold *Thresh* and a set of keywords relevant to its interests to its neighbors in the transmission range. When a node receives the Query message, it first calculates the node's interest in the query string. The files downloaded due to the querying behaviors in the past, and therefore reflect a user's interests and requesting patterns and it can be used to predict the requesting behaviors in the future. Thus, the latest files in the repository can be used to reflect the recent interest and requesting pattern. We define the *interest* value as the average of the similarities between the query string and the latest *r* downloaded files, where *r* is a pre-defined value. The formal definition of interest of node *N* for a query string *Q* is defined as:

$$int(N,\ Q) = \frac{\sum_{i=1}^{r} s_i(Q,\ f_i)}{r}$$

where *N* is the identifier of the node, *Q* is a query string in Query message, f_i is ith latest file, s_i is the similarity between query string *Q* and file f_i, and *r* is the number of files used to calculate the interest. We use *cosine distance* [10] to measure the similarity. The cosine distance is a widely used distance measure in information retrieval (IR) applications, and has been found historically to be quite effective in practical IR experiments. The similarity between query string *Q* and file *D* is defined as:

$$s(Q,\ D) = \frac{\sum_{k=1}^{NK} q_k d_k}{\sqrt{\sum_{k=1}^{NK} q_k^2 \sum_{k=1}^{NK} d_k^2}}$$

where *NK* is the number of keywords in query string *Q*, q_k is always 1, and d_k is 1 if the kth keyword in the query string *Q* exists in the meta-information of the file *D*, else d_k is 0.

If the interest of the node *int*(*N*, *Q*) is larger than the *Thresh*, the node forwards the Query message by broadcasting. If there are files matching the query in the node's repository or in the reference cache, the node returns a QueryHit message containing the references to the matching files along the query path to the *source node*, the node issuing the Query message and every node on the query path receiving QueryHit messages. Note that a file matches a query only if it matches all keywords in the query. When the nodes receive the QueryHit message, if the same entries for the references message exist in the reference cache, the Access_Time of the entries are updated, else the nodes create new entries for the QueryHit message. When the QueryHit messages reach the source node, it gets knowledge of the locations of the files and launches connections and file transmissions. Fig. 1 is an example using SLP to locate files.

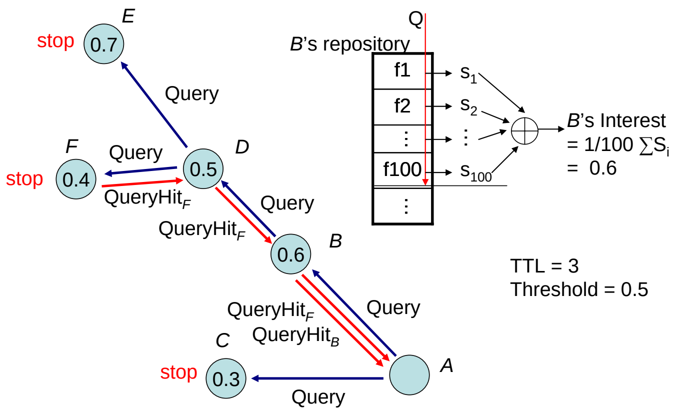


Fig. 1. Simple lookup protocol.

In Fig. 1, we assume TTL is equal to 3, and threshold is 0.5. The number on the node means its interest, QueryHit$_B$ means the QueryHit message from node *B*, and QueryHit$_F$ means the QueryHit message from node *F*. Node *A* initiates the lookup, and broadcasts a Query message. When node *A*'s neighbors *B* and *C* receive the Query message, they calculate their interest, and search for the files matching the query string in the repository and in the reference cache. In this example, because the interest of node *C* is 0.3 less than threshold 0.5, node *C* stops forwarding. On the other hand, because the interest of node *B* is 0.6 larger than the threshold, node *B* keeps forwarding the Query message to node *D*. Besides, there is a matching file reference in the reference cache, therefore node *B* returns a QueryHit message to source node *A*. Since there are no files matched in node *D*, it just forwards Query message to nodes *E* and *F*. There is a file matched in the repository of node *F* and node *F* returns a QueryHit message backwards along the path the Query message conveyed, and nodes *D* and *B* add the file information into their reference caches when receiving the QueryHit message. Node *F* stops forwarding because of insufficient interest value and TTL. Node *E* also stops forwarding because insufficient TTL. Finally, node *A* receives two references from nodes *B* and *F*, and launches connections to node *B* and/or node *F*, respectively.

The operation of SLP leads to a very attractive characteristic that the references to the files are implicitly replicated on the nodes which tend to request these files in the future. In this example, node *D* and node *B* which have at least 0.5 of interest replicate the references and they probably request the files later. This is because the QueryHit messages are returned backwards along the query path on which the interests of the nodes except for the end nodes are larger than the threshold value. All nodes on the query path replicate the references when receiving the QueryHit messages. Bandwidth efficiency in SLP is achieved by selective forwarding and further improved by high-utilization reference caches. Besides, SLP also avoids selfish behaviors because the behavior of forwarding query message is not only beneficial for the source node but also for the forwarding nodes themselves.

3.2 Advanced Lookup Protocol

Advanced lookup protocol (ALP) retains basic architecture of SLP and inherits all strengths from SLP such as reduction of messages, interest-based replication of references, and avoidance of selfish behaviors, but modifies the data structures and message formats and adds new features of incremental threshold and dynamic determining of TTL and threshold. In ALP, the initial threshold is lower and the probes at the beginning are extensive. The threshold is getting higher and the probes are narrowed down gradually. Therefore it will not give up too many possible resources at the beginning. Determination of TTL and threshold is dynamic with the peer-to-peer network condition such as the number of shared files in the network, the number of nodes in the network, *etc*.

Each mobile device in ALP maintains a *repository*, a *reference cache*, a *feedback table*, and a *helper table*. A repository stores the resources obtained from other peers in the local file system. A reference cache contains the information about the remote resources. An entry in the reference cache is composed of four fields: File_Ref, Interest, Popularity and Access_Time. A File_Ref consists of the pair of File_Src and File_Info. A File_Src field indicates the location information of a resource such as IP or MAC address of a mobile device. A File_Info field, which could be file name or meta-descriptions of the file, is used for query matching. The value of the Interest field is calculated when receiving the Query message and is filled in cache entry when receiving the QueryHit message. We expect that the higher the Interest is, the more likely the reference is to be used in the future. The Popularity indicates how popular the resource is. The Access_Time indicates the last time the reference entry is accessed.

TTL	5	5	5	5	5	···	1	1	1	1	1
Thresh	1.0~0.9	0.9~0.8	0.8~0.7	···	0.1~0.0	···	1.0~0.9	0.9~0.8	0.8~0.7	···	0.1~0.0
N_FF	4.12	6.3	8.25	···	36.1	···	0.02	0.06	0.11	···	0.3

Fig. 2. Feedback table.

A feedback table contains three fields: TTL, Threshold, and N_FF. The feedback table is used to keep track of historical query results for estimating the most appropriate TTL and threshold for the lookup. It shows that how many files could be found under what kind of TTL and threshold assigned to a query.

For example, an entry of <5, 0.8~0.7, 8.25> means that a query with TTL equal to 5 and threshold ranging between 0.8 and 0.7 are able to get 8.25 results on the average. We divide the range of threshold into ten sections, (1.0~0.9), (0.9~0.8), …, and (0.1~0.0). Then the number of entries of the feedback table is $TTL_{max} \times 10$, where TTL_{max} is maximum value of TTL allowed in the system. In this example, the TTL_{max} is five, and the size of feedback table is fifty. The size of feedback table is independent of the number of nodes.

The helper table is used to keep track of the relationship between a Query message and its corresponding QueryHit messages to help maintenance of the reference cache and the feedback table. The entry of the helper table is composed of five fields: Query_ID, TTL, Thresh, Interest, and N_FF. When a node receives the Query message, it adds an entry and copies the Id_{Query}, TTL, threshold Thresh of the Query message and its interest in the query string to the Query_ID field, TTL field, Thresh field, and Interest field of the new entry, respectively. The default value of N_FF is zero. The N_FF means the number of files found and it is accumulated whenever the node receives the corresponding QueryHit messages. Every entry of the helper table lives for a predefined period of time. Before being removed, the information of the entry is transferred in the form of <TTL, Thresh, N_FF> to the feedback table. The new N_FF value of the feedback entry is updated as 1/2 N_FF_{helper} + 1/2 $N\_FF_{feedback}$, where N_FF_{helper} is N_FF in helper table and reflects the current network condition, $N\_FF_{feedback}$ is N_FF in feedback table and reflects the historical information.

We modify the formats of Query message and QueryHit message defined in SLP. A Query message in ALP is a query string Q including one or more keywords, TTL, a threshold Thresh, an increment Inc, and a unique identifier Id_{Query} for the Query message. TTL is the maximum number of hops a message can be forwarded. Threshold Thresh is a minimum interest value for query forwarding. Id_{Query} is chosen by hashing the combination of the address of the source node, the query string, and the timestamp. The Inc is the unit of threshold increment. A QueryHit message consists of file references, hop count Hops for retrieving the files, and the identifier Id_{Query} of the Query message it corresponds to. The initial value of Hops is one, and is increased by one as the QueryHit message is forwarded. The Id_{Query} is retrieved from the received Query message.

Besides, a system parameter PNFF (prospective number of file found) can be set to promise around PNFF files will be found. A larger value of PNFF leads to more but slower responses. Contrarily, a smaller value leads to faster but fewer responses. After PNFF is set, when someone desires to issue queries, the most appropriate TTL and Thresh are determined dynamically for the query request. If the query hit h files in the reference table, we update PNFF to PNFF – h. We search for two adjacent entries E_1 = <TTL_1, $Thresh_1$, N_FF_1> and E_2 = <TTL_2, $Thresh_2$, N_FF_2> in the feedback table such that TTL_1 is the same as TTL_2, N_FF_1 is smaller than PNFF, and N_FF_2 is larger than PNFF. Finally, we calculate the threshold *Thresh* by the ratio of equality:

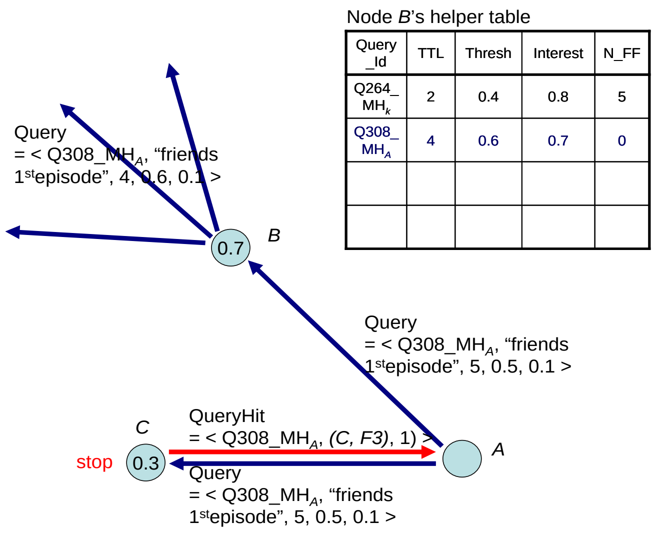
$$\frac{\text{PNFF} - \text{N\_FF}_1}{\text{PNFF} - \text{N\_FF}_2} = \frac{\text{Thresh} - \text{Thresh}_1}{\text{Thresh} - \text{Thresh}_2}$$

$$\Rightarrow \text{Thresh} = \frac{\text{Thresh}_1(\text{PNFF} - \text{N\_FF}_2) - \text{Thresh}_2(\text{PNFF} - \text{N\_FF}_1)}{\text{N\_FF}_1 - \text{N\_FF}_2}.$$

After executing the above steps, the TTL and Thresh are produced and filled in the Query message. The increment Inc is set to (*MaxThresh-Thresh*)/*TTL*, where *MaxThresh* is the maximum threshold allowed in the network. If there are not enough information for calculating TTL and Thresh, we just set TTL to maximum TTL allowed in the system and Thresh to zero.

After calculation of TTL, Thresh and Inc, the source node broadcasts the Query message to their neighbors in the transmission range. When a node *N* receives the Query message, it searches for files matching the query in the repository or in the reference cache and calculates the node's interest in the query message. If the interest of the node *int*(*N*, *Q*) is larger than or equal to the threshold, the node keeps track of the information <Id_{Query}, *TTL*, *Thresh*, *int*(*N*, *Q*), *N_FF*> in the helper table, and forwards the Query message by broadcasting. The field Thresh of the forwarded Query message is increased by Inc. If there are files matching the query in the reference cache, the node returns a QueryHit message and increases the Popularity value of the corresponding cache entry. A QueryHit message contains the identifier of corresponding Query message, a hop count initial to one and the references of the files matching the query backwards along the query path to the source node.

When a node receives the QueryHit message, if the same entries for the QueryHit message exist in the reference cache, the Popularity field and Access_Time field of the entries are updated; otherwise the node creates new entry of the reference cache for the QueryHit message. The content of File_Ref field of the created entry of the reference cache is retrieved from the reference in the QueryHit message. The value of Interest field of the created entry of reference cache is retrieved from the entry of the helper table in which the identifier of the Query message is equal to the identifier in the QueryHit message. The Popularity is zero and the Access_Time is the time the entry is created. The entries of the helper table live for a period of time for helping collect query results. Before its end, the query result (the number of the files found) is transferred to the N_FF field of corresponding entry in the feedback table. When the QueryHit messages reach the source node, it can get knowledge of the location of the files and launch connections and transmissions. Fig. 3 is an example shows how ALP works.

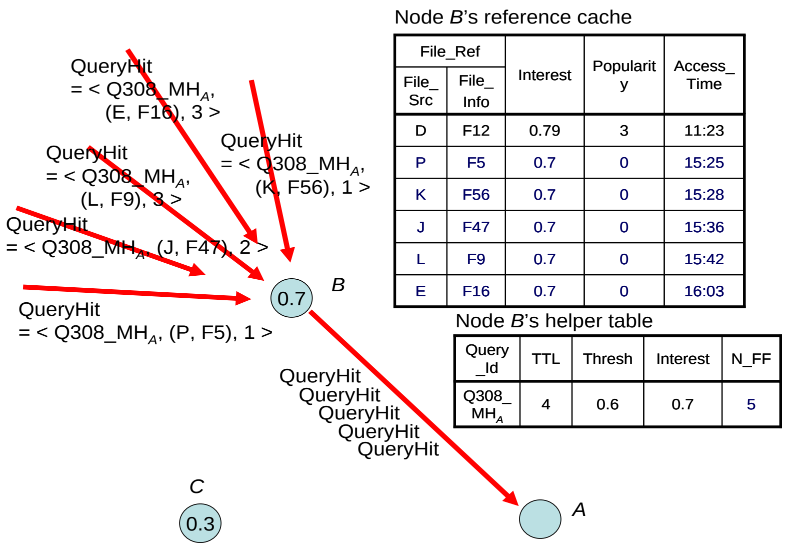

Node *B*'s helper table
Query = < Q308_MH$_A$, "friends 1stepisode", 4, 0.6, 0.1 >
0.7
B
Query = < Q308_MH$_A$, "friends 1stepisode", 5, 0.5, 0.1 >
C
QueryHit = < Q308_MH$_A$, *(C, F3)*, 1) >
stop
0.3
A
Query = < Q308_MH$_A$, "friends 1stepisode", 5, 0.5, 0.1 >

Query_Id	TTL	Thresh	Interest	N_FF
Q264_MH$_k$	2	0.4	0.8	5
Q308_MH$_A$	4	0.6	0.7	0

Fig. 3. Advanced lookup protocol – receiving Query.

In Fig. 3, node *A* broadcasts a Query message where $\mathrm{Id}_{\mathrm{Query}}$ is Q308_MH$_A$, keywords in the query string Q are “friends” and “1st episode”, TTL is 5, and threshold Thresh is 0.5, and Inc is 0.1. Assume that node *A*’s neighbors nodes *B* and *C* receive the Query message. Because node *B*’s interest is 0.7 larger than threshold, node *B* forwards the Query message with new TTL = 4, and increases threshold to 0.6. Besides, node *B* creates a new entry <Q308_MH$_A$, 4, 0.6, 0.7, 0> in the helper table. The information contained in the helper table is used for the maintenance of the reference cache and the feedback table. On the other hand, node *C* returns a QueryHit but stops forwarding because there are files matched in node *C* but node *C*’s interest is 0.3 less than the threshold. The QueryHit message consists of an identifier copied from the corresponding Query message, a hop count 1, and a file reference (*C*, *F*3) where *C* indicates the file source, and *F*3 is the information for the file.

In Fig. 4, node *B* receives the responses of the Query message but node *C* doesn’t receive any response because it didn’t forward the Query message due to its insufficient interest. When node *B* receives the QueryHit messages, it adds entries for them in the reference cache. For example, the cache entry <*P*, *F*5, 0.7, 15:25> is added for the QueryHit message <Q308_MH$_A$, (*P*, *F*5), 1>. Besides, node *B* returns the QueryHit message to node *A*. Finally, source node *A* gets five query results. In our protocol, we do not record the hop counts for the files in the reference cache, because the nodes are able to move arbitrarily in the mobile environment and lead to the recorded hop counts may not make sense when requesting the file.



QueryHit = < Q308_MH$_A$, (E, F16), 3 >
QueryHit = < Q308_MH$_A$, (L, F9), 3 >
QueryHit = < Q308_MH$_A$, (K, F56), 1 >
QueryHit = < Q308_MH$_A$, (J, F47), 2 >
QueryHit = < Q308_MH$_A$, (P, F5), 1 >
B
0.7
QueryHit
QueryHit
QueryHit
QueryHit
QueryHit
C
0.3
A

Node *B*’s reference cache

File_Ref		Interest	Popularity	Access_Time
File_Src	File_Info			
D	F12	0.79	3	11:23
P	F5	0.7	0	15:25
K	F56	0.7	0	15:28
J	F47	0.7	0	15:36
L	F9	0.7	0	15:42
E	F16	0.7	0	16:03

Node *B*’s helper table

Query_Id	TTL	Thresh	Interest	N_FF
Q308_MH$_A$	4	0.6	0.7	5

Fig. 4. Advanced lookup protocol – receiving QueryHit.

3.3 Maintenance of Data Structures

In SLP, every node needs to maintain two data structures, which are a repository and a reference cache. The repository is managed by the local file system. When the node retrieves a file, it is stored in the repository. The files in the repository also can be deleted by the user when unnecessary. As to the reference cache, it contains a fixed number of entries. The entry is added when the node receives a new file reference. When the space of the cache is inadequate, we use LRU (least recently used) algorithm [24] for replacement.

In ALP, every node needs to maintain four data structures, which are a repository, a reference cache, a helper table, and a feedback table. The maintenance of the repository is the same as those in SLP, but the maintenance of the reference cache is a bit different from those in SLP. We combine LRU algorithm and second-chance algorithm to develop a new algorithm for cache replacement. We use LRU as basic replacement algorithm but give most popular and most interested references second chances. When an entry has been selected, we inspect whether it has the right of second chance. If it does not have the right of second chance, we proceed to replace this entry. If it has the right, we give that entry a second chance and move on to select the next LRU entry. When an entry gets a second chance, its Access_Time is updated to the current time. As to the helper table, an entry is created when nodes receives a Query message, and lives for a predefined period of time. Before it is removed, the N_FF value of the entry is transferred to the feedback table. The size of a feedback table is fixed and independent of the number of the nodes on the network. When the latest information N_FF stored in the helper table is transferred to the feedback table, the N_FF in the feedback table is updated to reflect the latest network condition.

4. SIMULATION RESULTS

In this section, we demonstrate the simulation results comparing the performance of our protocols *simple lookup protocol* (*SLP*) and *advanced lookup protocol* (*ALP*) with *Passive Distributed Indexing* (*PDI*) [15], a lookup protocol for the mobile environment. The mobile devices are randomly placed within an area of 1,000 meters $\times$ 1,000 meters. There are 100 mobile devices moving according to the random waypoint mobility model [1] in this area. The transmission range for each mobile device is 150 meters. The speed of the device is randomly chosen from 0 to 1.5 meters per second. Each device maintains a repository which is able to store 512 files and a reference cache with capacity 32 entries. We assume each device initially stores 100 files in the local repository.

For the file matching, we assume files and query strings are composed of several keywords chosen from the keyword set K of size $N_{keywords} = 256$. $N_{interests}$ is number of keywords a node is interested in. For each node, $N_{interests}$ keywords can be randomly chosen from the keyword set K to form the initial files and the query strings. We assume each file has 8 to 16 attributes, which implies that each file is composed of 8 to 16 keywords. The query string consists of at most 3 keywords. To calculate the interest, we average the similarities of the latest 10 files in the repository. The *time-to-live* (*TTL*) for query messages is 3. Interest threshold for SLP is 0.2. The initial interest threshold and the maximum interest threshold for ALP are 0.0 and 0.5, respectively. We simulate 5000 query requests issued by randomly chosen peers.

We make our observations from six performance measures: success rate, scalability, bandwidth efficiency, query efficiency, search responsiveness, and search efficiency.

Finally, we demonstrate our ALP with supporting dynamic determining TTL and threshold for query requests.

(A) *Success rate*: In this experiment, we investigate the success rates of our protocols and PDI is also involved for comparison purpose. We say a query request is successful if only if it finds at least one replica of the file. The success rate is defined as:

$$success\ rate = \frac{number\ of\ successful\ query\ requests}{number\ of\ all\ query\ requests}.$$

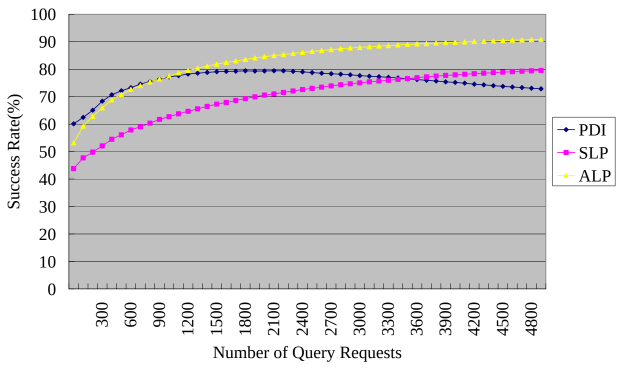


Fig. 5. Number of query requests vs. success rate.

The simulation result is shown in Fig. 5. The success rates of SLP and ALP increase gradually, and reach about 80% and 90% at the end of the simulation. With the growing number of query requests, the number of files to be spread, replicated, and shared is increasing. Therefore, it becomes easier to find matching files, and the success rate grows steadily. We also find an interesting result that the success rate of PDI reaches its peak 80% at executing around 2,000 queries, and then falls. Since PDI use local broadcast of query messages and response messages, its success rate booms at the beginning. With the growing number of shared resources, broadcasting query responses will fill caches with junk entries, and it also lead to the replacement of entries frequently.

(B) *Scalability*: In this experiment, we examine the scalability of the protocols. A scalable lookup protocol should involve only few messages during the search process. We exploit *Messages per Query* to evaluate the scalability of lookup protocols.
As shown in Fig. 6, on average, there are 46.97 messages generated during a PDI search process, 13.84 messages during SLP, and 24.79 messages during ALP. We raise the scalability up to 339%. Only few messages generated by SLP, since query messages are selectively forwarded, and query hits are backwards returned. ALP generates slightly more messages due to its incremental-threshold policy. PDI generates much more messages than SLP and ALP, since it locally broadcasts not only query messages but also response messages. Therefore, our protocols are more scalable.

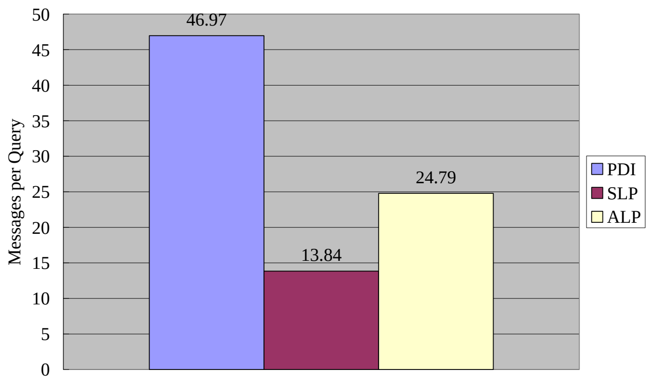


Fig. 6. Messages per query.

(C) *Bandwidth Efficiency*: In this experiment, we investigate the efficiency of bandwidth of the protocols. A bandwidth-efficient lookup protocol should take only few messages to achieve a successful query request. Therefore, *Messages per Success* (*Cost of Success*) is used to measure the bandwidth efficiency. The simulation result is shown in Fig. 7.

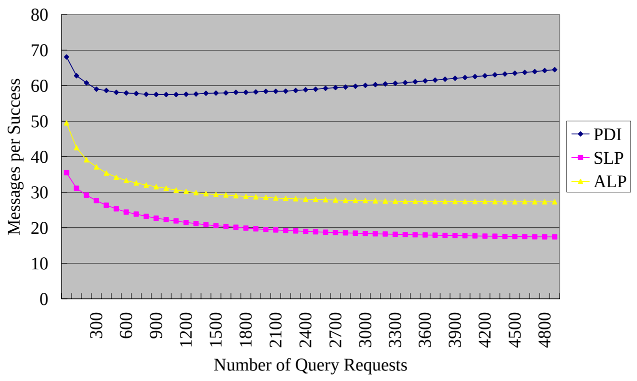


Fig. 7. Number of query requests vs. messages per success.

Before executing around 1,200 queries, the replicated files and references of our schemes help raise the probability of success and reduce the required messages remarkably. Due to the limited size of the cache, afterwards, the cache reaches full utilization, replacement for cache entries starts to take place frequently, and reduction of the cost of success becomes slower. We also find an interesting phenomenon that the cost of success of PDI even raises after executing around 1,200 queries. The reason is that local broadcasting makes query responses overflowing, caches full of junk entries, the replacement of entries quite frequently, and utilization of cache inefficient. Since our protocols take only few messages and achieve satisfactory success rates, from the simulation result, we can obviously observe that SLP and ALP are much bandwidth-efficient than PDI.

(D) *Query Efficiency*: Query Efficiency [14] is defined as the ratio of query hits of messages per node:

$$Query\ Efficiency = \frac{Nubmer\ of\ QueryHits}{\dfrac{Number\ of\ Messages}{Number\ of\ Nodes}} = \frac{Number\ of\ QueryHits}{Messages\ per\ Node}.$$

Our interest-based selective forwarding scheme reduces number of messages effectively and still has good search performance. From Fig. 8, we can observe that Query Efficiencies of our protocols SLP and ALP are 3.5 and 2.5 times higher than PDI. We also find that Query Efficiency of PDI remains almost constant after executing around 1,200 queries. The reason is that local broadcasting makes query responses overflowing, caches full of junk entries, and therefore the searching inefficient.

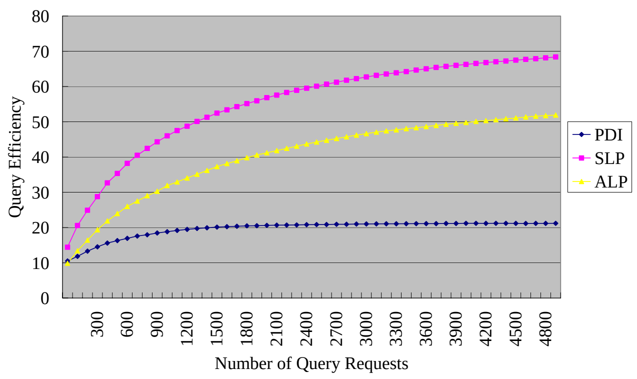


Fig. 8. Number of query requests vs. query efficiency.

(E) *Search Responsiveness*: Search Responsiveness [14] evaluates responsiveness and reliability. Responsiveness is the ability of a lookup protocol to respond quickly to meet the needs of a user. Reliability means the ability for a lookup protocol to find out the matching resources successfully. Therefore, Search Responsiveness measuring the responsiveness and reliability of a lookup protocol can be defined as:

$$Search\ Responsiveness = \frac{Success\ Rate}{Average\ Path\ Length}.$$

The simulation result is shown in Fig. 9. Because of the efficient cache utilization, our protocols have higher success rate and lower path length. We also find that the method of incremental threshold of ALP improves the responsiveness further. On the other hand, local broadcasting scheme makes cache utilization of PDI inefficient, and the success rate decays after running for a period of time (as shown in Fig. 5). Therefore, our protocols are more responsive than PDI.

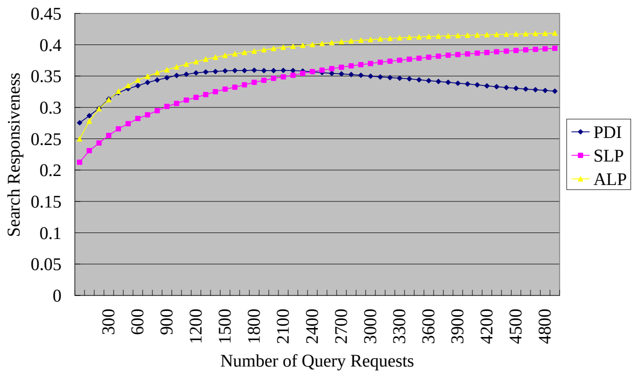


Fig. 9. Number of query requests vs. search responsiveness.

(F) *Search Efficiency*: In this experiment, we measure the overall performance of a lookup protocol. A unified criterion Search Efficiency [14] can be defined as:

$$\textit{Search Efficiency} = \textit{Query Efficiency} \times \textit{Search Responsiveness}.$$

The simulation result is shown in Fig. 10. Since the Query Efficiency (see Fig. 8) and Search Responsiveness (see Fig. 9) of our protocols are much better than those of PDI, our protocols outperform PDI by up to 385%.

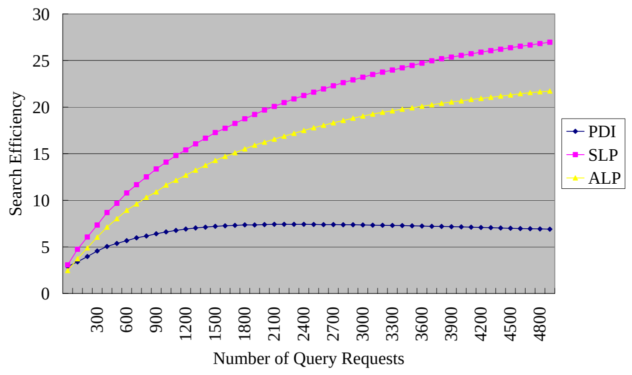


Fig. 10. Number of query requests vs. search efficiency.

5. CONCLUSIONS

Lookup protocol is one of the critical issues for peer-to-peer networks. However, the most of existing protocols are not suitable for a mobile environment. In this paper, we propose two protocols, SLP and ALP, for MANETs. We use interest-based selective forwarding mechanisms to reduce the overhead on the networks and the operation of our protocols also leads to a very attractive characteristic that the references to the files are implicitly replicated on the nodes which tend to request these files in the future. Therefore, when a node issues a query, some file references are probably hit in the local cache, and the query message to be sent can be with a smaller TTL and a larger threshold to reduce the overhead further. We also propose a scheme, incremental threshold, to improve the performance further. The initial threshold is lower and the probes at the beginning are extensive. The threshold is getting higher and the probes are narrowed down gradually. Therefore it will not give up too many possible resources at the beginning. We measure the lookup protocols from six aspects: success rate, scalability, bandwidth efficiency, query efficiency, search responsiveness, and search efficiency. Simulation results show that our protocols have higher success rate and raise the scalability and bandwidth efficiency comparing to PDI. Query efficiency and search efficiency are also better than PDI. Besides, our protocols are more responsive. Finally, our protocols can avoid selfish behaviors, since the behavior of forwarding queries benefits not only the source node but also the forwarding node.

REFERENCES

1. J. Broch, D. Maltz, D. Johnson, Y. C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1998, pp. 85-97.
2. I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: a distributed anonymous information storage and retrieval system," in *Proceedings of the International Computer Science Institute* (*ICSI*) *Workshop on Design Issues in Anonymity and Unobservability*, 2000, pp. 311-320.
3. S. Daswani and A. Fisk, "Gnutella UDP extension for scalable searches v0.1," http://cvs.limewire.org/.
4. eDonkey, http://www.edonkey2000.com/.
5. eMule, http://www.emule-project.net/.
6. F. P. Franciscani, M. A. Vasconcelos, R. P. Couto, and A. A. F. Loureiro, "Peer-to-peer over ad hoc networks: (re)configuration algorithms," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003, pp. 32-41.
7. Gnutella, "The Gnutella protocol specification v0.4," http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
8. Gnutella2, "The Gnutella2 protocol specification," http://www.gnutella2.com/index.php/Main_Page.
9. Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: peer-to-peer patching scheme for VoD service," in *Proceedings of the International Conference on World Wide Web*, 2003, pp. 301-309.
10. D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*, MIT Press, Cambridge, MA, U.S.A., 2001.
11. C. E. Perkins, *Ad Hoc Networking*, Addison-Wesley, New York, 2001.
12. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 654-663.
13. KaZaA, http://www.kazaa.com/.

14. T. Lin and H. Wang, "Search performance analysis in peer-to-peer networks," in *Proceedings of the 3rd International Conference on Peer-to-peer Computing*, 2003, pp. 204-205.
15. C. Lindemann and O. P. Waldhorst, "A distributed search service for peer-to-peer file sharing in mobile applications," in *Proceedings of the 2nd International Conference on Peer-to-Peer Computing*, 2002, pp. 73-80.
16. Morpheus, http://www.morpheus.com/.
17. Napster, http://www.napster.com.
18. A. Oram, *Peer-to-Peer: Harnessing the Benefits of a disruptive Technology*, O'Reilly and Associates, Inc., Sebastopol, CA, U.S.A., 2001.
19. M. Papadopouli and H. Schulzrinne, "Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2001, pp. 117-127.
20. G. P. Premkumar, "Alternate distribution strategies for digital music," *Communications of the ACM*, Vol. 46, 2003, pp. 89-95.
21. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM*, 2001, pp. 161-172.
22. M. Rieanu, "Peer-to-peer architecture case study: Gnutella network," in *Proceedings of the 1st International Conference on Peer-to-Peer Computing*, 2001, pp. 99-100.
23. A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, 2001, pp. 329-350.
24. A. Silberschatz, G. Gagne, and P. B. Galvin, *Operating System Concepts*, 6th ed., John Wiley and Sons, Inc., New York, 2002.
25. K. Sripanidkulchai, B. Maggs, H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *Proceedings of the IEEE INFOCOM*, 2003, pp. 53-63.
26. I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," *IEEE/ACM Transactions on Networking*, Vol. 11, 2003, pp. 17-32.

Yi-Chung Chen (陳貽中) received the B.S. degree in Computer Science and Engineering from Yuan Ze University, Chungli, Taiwan, R.O.C., in 2002 and the M.S. degree in Computer Science and Information Engineering from National Central University, Chungli, Taiwan, R.O.C., in 2004, respectively. He received the High Honor Paper Award of the 19th IEEE International Conference on Advanced Information Networking and Applications. His current research interests include quality of service (QoS) in wireless and mobile networks, peer-to-peer computing, and pervasing computing.

Jang-Ping Sheu (許健平) received the B.S. degree in Computer Science from Tamkang University, Taiwan, R.O.C., in 1981, and the M.S. and Ph.D. degrees in Computer Science from National Tsing Hua University, Taiwan, R.O.C., in 1983 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, R.O.C., as an Associate Professor in 1987. He is currently a Professor of the Department of Computer Science and Information Engineering and Director of Computer Center, National Central University. He was a Chair of Department of Computer Science and Information Engineering, National Central University from 1997 to 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine from July 1999 to April 2000. His current research interests include wireless communications, mobile computing and parallel processing. He was an associate editor of Journal of the Chinese Institute of Electrical Engineering, from 1996 to 2000. He was an associate editor of Journal of Information Science and Engineering from 1996 to 2002. He was an associate editor of Journal of the Chinese Institute of Engineers from 1998 to 2004. He is an associate editor of IEEE Transactions on Parallel and Distributed Systems. He was a Guest Editor of Special Issue for Wireless Communications and Mobil Computing Journal. He was a Program Chair of IEEE ICPADS 2002. He was a Vice-Program Chair of ICPP 2003.

He received the Distinguished Research Awards of the National Science Council of the Republic of China in 1993-1994, 1995-1996, and 1997-1998. He was the Specially Granted Researchers, National Science Council, from 1999 to 2005. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He received the Distinguished Professor award of the National Central University. Dr. Sheu is a senior member of the IEEE, a member of the ACM, and Phi Tau Phi Society.