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, *Taiwan, R.O.C.*
s1522067@cc.ncu.edu.tw sheujp@csie.ncu.edu.tw

Abstract

In this paper, we propose two 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.

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. 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. A mobile ad hoc network (MANET) consists of mobile devices communicating with each other using multi-hop wireless links without any central control. 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 mobileenvironments is still a new research issue. 7DS [6] is the first approach to apply peer-to-peer technique to mobile environments. For queries, 7DS implements a multi-hop flooding algorithm combined with multicast delivery of queries. PDI [5] 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 MANETs. In contrast to the previous work PDI, 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 protocol such as *DSR* [3], *DSDV* [7], and *AODV* [8].

The rest of this paper is organized as follows. Section 2 presents our lookup protocols. In Section 3, we evaluate the performance of our protocols through simulations. Finally, we conclude our contributions in Section 4.

2. 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. 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. Efficiency of bandwidth is improved further in ALP by referring to feed-

This work was supported by the National Science Council of the Republic of China under grants NSC 92-2213-E-008-006 and NSC 93-2752-E-007-003-PAE.

backs from previous query requests. We present the details of two protocols in the following.

2.1. Simple Lookup Protocol (SLP)

Since traditional flooding mechanism is not bandwidth-efficient, we propose interest-based lookup protocols 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 i^{th} 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* [2] 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. 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. 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.

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. 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.

2.2. Advanced Lookup Protocol (ALP)

Advanced lookup protocol 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. The File_Ref and Access_Time are same as in the SLP. 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.

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} x 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, Hops (hop count 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. 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 and 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. Figure 3 is an example shows how ALP works.

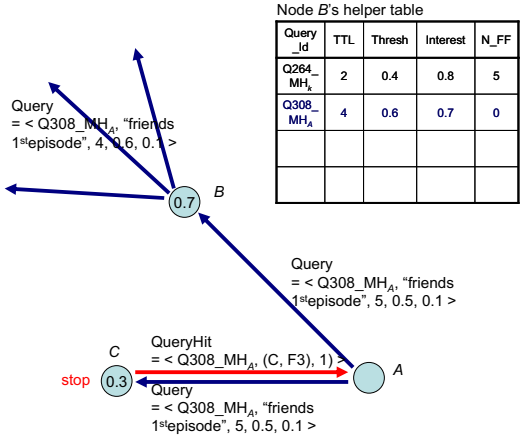

Figure 1. Advanced Lookup Protocol – Receiving Query

In Figure 1, node *A* broadcasts a Query message where Id_{Query} is $Q308\_MH_A$, keywords in the query string Q are "friends" and "1st episode", TTL is 5, 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 Figure 2, 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.

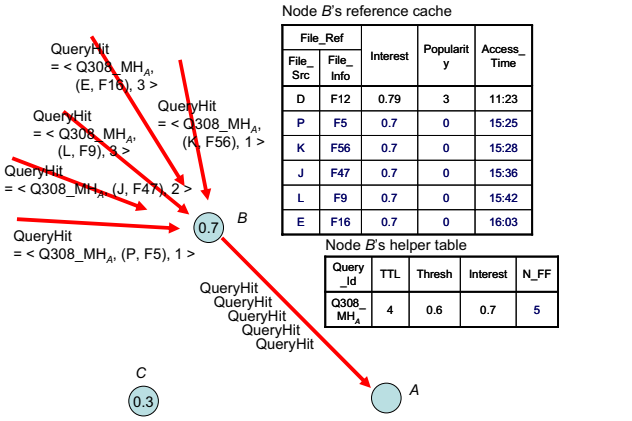

Figure 2. Advanced Lookup Protocol – Receiving QueryHit

2.3. 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 [9] 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 [9] and second-chance algorithm [9] 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.

3. Simulation Results

In this section, we demonstrate the simulation results comparing the performance of our protocols *SLP* and *ALP* with *PDI* [5], a lookup protocol for the mobile environment. The mobile devices are randomly placed within an area of 1000 m × 1000 m. 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 m. The speed of the device is randomly chosen from 0 to 1.5 m 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. 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 four performance measures: success rate, scalability, bandwidth efficiency, and search responsiveness.

(A) *Success rate*: 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:

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

The simulation result is shown in Figure 3. 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 2000 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.

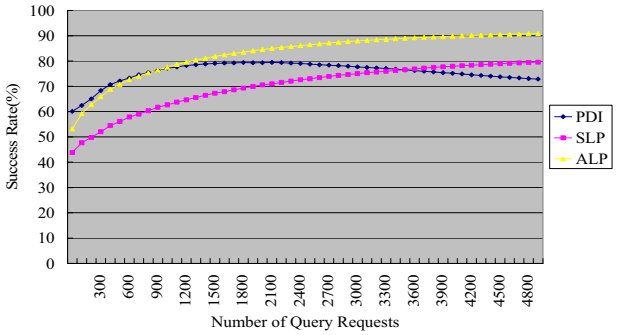

Figure 3. Number of Query Requests vs. Success Rate

(B) *Scalability*: A scalable lookup protocol should involve only few messages during the search process. Hence, we exploit *Messages per Query* to evaluate the scalability of lookup protocols.

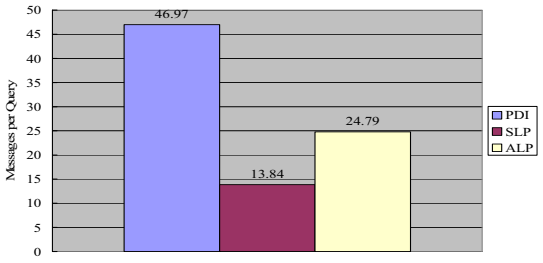

Figure 4. Messages per Query

As shown in Figure 4, on average, there are 46.97 messages generated during a PDI search process, 13.84 messages during SLP, and 24.79 messages during ALP. 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.

(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 simu-

lation result is shown in Figure 5. Before executing around 1200 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. From the simulation result, we can observe that SLP and ALP are much bandwidth-efficient than PDI.

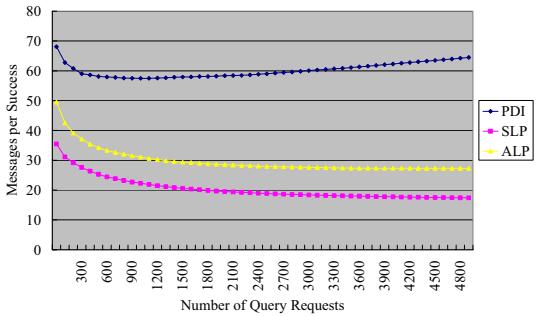

Figure 5. Number of Query Requests vs. Messages per Success

(D) *Search Responsiveness*: Search responsiveness [4] 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 Figure 6. 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 Figure 3). Therefore, our protocols are more responsive than PDI.

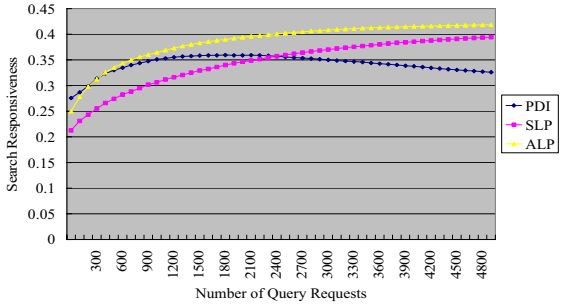

Figure 6. Number of Query Requests vs. Search Responsiveness

4. Conclusions

Lookup protocol is one of the critical issues for peer-to-peer networks. However, the most of existing protocols are not suitable for mobile environments. 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 implicitly replicate files on the nodes which tend to request these files in the future. We also propose a scheme, incremental threshold, to improve the performance further. 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. Simulation results show that our protocols have higher success rate and raise the scalability and bandwidth efficiency comparing to 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 nodes.

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*, pp. 85-97, 1998.

[2] D. Hand, H. Mannila, and P. Smyth, "Principles of Data Mining," MIT Press, 2001.

[3] D. B. Johnson, D. A. Maltz, and J. Broch, "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks" in *Ad Hoc Networking*, Edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001.

[4] 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*, pp. 204-205, 2003.

[5] 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*, pp. 73-80, 2002.

[6] 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*, pp. 117-127, 2001.

[7] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *Proceedings of ACM SIGCOMM'94*, pp. 234-244, 1994.

[8] C. E. Perkins and E.M. Royer, "Ad Hoc On-Demand Distance Vector Routing," in *Proceedings of the 2nd*

IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100, 1999.

[9] A. Silberschatz, G. Gagne, and P. B. Galvin, "Operating System Concepts," John Wiley & Sons, Inc., 6th Edition, 2002.