

An Efficient Update Algorithm for Supporting Mobility in Structured P2P Systems

Boxuan Gu

Department of Computer Science and Engineering
Ohio State University, Columbus, OH 43210

Hu Wang

Department of Computer Science
University of Kentucky, Lexington, KY 40506

Zongming Fei

Abstract—Structured peer to peer (P2P) systems organize a large number of nodes in well-defined structures and provide efficient and scalable search functionalities. They have been used as a basis to implement many network services and applications. As more and more computers have wireless connections and can move from one place to another, supporting mobility in the structured P2P systems becomes an important issue. In this paper, we present an algorithm called Range based Updating Algorithm (RUAM) to efficiently support mobile nodes in structured P2P systems. RUAM allows a mobile node to actively update information stored in other nodes when it changes its address. Hence, the outdated routing information in the system can be greatly decreased. The simulation results show that RUAM is efficient in updating stale states caused by mobile nodes and greatly improves the lookup performance.

Keywords: Mobility, Peer to Peer, DHT, overlay networks

I. INTRODUCTION

Peer to peer (P2P) computing relies on the cooperation among peers¹ to accomplish system tasks and is a paradigm shifted from the traditional client/server model for information exchange. Instead of depending on a central server, a peer can retrieve information and get service from other peers. Because storage, processing and retrieval are distributed to all the peers, peer to peer computing removes the potential bottleneck in the client/server model and has better scalability for building large scale systems. It has been used as a fundamental structure to implement many network services and applications, such as massive storage systems [1], multicast, event-notification [2], security [3], information retrieval and publish-subscribe systems.

A structured P2P system organizes participating nodes into a logical topology [4], [5], [6], [7]. Each node is assigned a unique logical identification (ID) and has a routing table that consists of information about a set of other peers. The node can directly contact the peers in its routing table. An object in the system also has a unique ID. It is usually stored in the peer whose ID is the closest to the ID of the object. This peer is called the home node or the root node of the object. To search an object in a structured P2P system, peers route a query in a greedy manner by choosing the next hop based on their routing tables and the object ID. Specifically, a peer chooses from its routing table a node whose ID is the closest to the object ID. In this way, the queries are routed

through the intermediate peers whose IDs are closer and closer to the root node until the root node is reached. Typically, a structured P2P system locates an object in $O(\log N)$ hops with only $O(\log N)$ or less entries in each routing table. Most proposed structured P2P systems assume participating nodes are stationary.

In the current Internet, however, we have witnessed that more and more computers have wireless connections and can move from one place to another. The mobile environment presents new challenges to the peer to peer computing. When a machine moves from one location to another, it may be out of service for a period of time. Even when the machine finally reconnects to the Internet at a new location, the network it connects to and its IP address will be different. A simple solution to the mobility problem in the P2P system is to treat the moving of a mobile node as the node leaving the system and its reconnecting with the system as a new join. Original join and leave protocols can be used for these processes. Obviously, this simple scheme will affect the functioning and performance of the P2P system. In particular, if the ID of a node is dependent on the IP address (by a hash function), the change of the IP address will lead to the change of the ID. Since the peering relationships in most structured P2P systems are based on the logical IDs, the change of the ID will subsequently lead to the node finding new neighbors. This causes the maintenance overhead. In addition, content stored at a node usually depends on the node's ID. Therefore, an address change will result in a significant amount of data being shipped to other nodes and new data being transferred to this node, and thus will consume a lot of network resources.

In this paper, we propose a new approach to support the mobility in P2P systems based on the separation of a node's ID and its IP address. Currently, the peering relationships are maintained by remembering the IP addresses of one's neighbors. Specifically, when node B is node A 's neighbor, A will store B 's IP address. In contrast, our solution suggests that the relationships are established purely based on logical IDs. If B is node A 's neighbor, A will store B 's logical ID and the related information about this ID, such as its state, its associated IP address. The association of these logical IDs to IP addresses is through *dynamic binding*. If all nodes in a P2P system are wired, the IDs stored at a peer will be bound with the unique IP address of each of its neighbors. The peer to peer system will function in a similar way as most current

¹Peers and nodes are used interchangeably in this paper.

P2P systems. In the mobile environment, some nodes in the P2P system are mobile, and they may have different IP addresses at different locations. In our solution, a node maintains its logical ID so that the content stored at this node does not have to be changed. The association of the ID with the new IP address must be updated to all its peers so that they can find this node at its new location.

To this end, we design an efficient algorithm, *Range-based Updating Algorithm* (RUAM), to perform such updating. The basic idea of RUAM is that a mobile node can determine several ranges in the identifier space that contain the nodes having pointers to it. We hereafter refer to these nodes as the *related nodes* of the mobile node. Then the mobile node finds an efficient way to locate the first node in the first range. From this node, it updates nodes and transfers the updating message from one range to another. The mobile node actively updates its related nodes when it is leaving or rejoining. This active updating helps maintain the consistency of the routing tables in the related nodes and improve the performance of the search process.

The rest of the paper is organized as follows. We present related work in section II. Section III illustrates our solution and RUAM in particular. We present the simulation results in section IV. Section V concludes the paper.

II. RELATED WORK

Mobility issues have been studied in the structured P2P systems. ROAM[10] aims to provide seamless mobility for the Internet users based on internet indirection infrastructure (i3) [11]. A mobile node registers with the i3 node which serves as a rendezvous node of the mobile node in the i3 layer. Messages for the mobile node are firstly delivered to the i3 layer. Then the rendezvous node routes the messages to the mobile node. When the mobile node moves, it updates the address in its rendezvous node. Similarly, in [12], Zhao *et al.* proposed WARP, a mobility infrastructure based on Tapestry[5], to support the mobility of nodes. Both ROAM and WARP assume a stationary layer to support the mobility and use the structured P2P system to provide mobility service. In contrast, we focus on the issues in the mobility of nodes within a P2P system itself, *i.e.*, how to accommodate mobile nodes as a part of the P2P system.

A similar problem has been addressed in [13]. A mobile node informs its neighbors of an appropriate node to replace itself when the mobile node starts moving. When it returns, the mobile node uses the join procedure to rejoin P2P systems with a new IP address. Joining as a new node is costly for a mobile node due to message exchanges in building the finger table, updating states in other nodes, and transferring keys. Our scheme avoids these problems by having the mobile node rejoin the system with the same ID it has before. The RUAM algorithm efficiently updates those nodes pointing to the ID with the new IP address.

Bristle is proposed as a mobile structured P2P system [14]. Similar to i3 [11], Bristle contains two layers,

a stationary layer and a mobile layer. Unlike i3 [11], a mobile node pushes queries into the stationary layer only if it cannot route queries further by using its routing table. Both layers interleaves together to form a P2P system. However, Bristle still distinguishes participating nodes as stationary nodes and mobile nodes. On the contrary, our work accommodates both stationary nodes and mobile nodes as a single system and can make the routing more efficient.

Also related is mobile IP [9], an extension of normal IP protocol. It can possibly be used to support mobility for peer to peer systems. In mobile IP, the node that wants to communicate with a mobile node is called a *correspondent node*. It can talk to the mobile node through the mobile node's home agent by triangular routing. However, this is not as efficient as using the shortest path from the correspondent node to the mobile node directly. This problem is addressed in mobile IP by introducing several messages exchanged between the home agent and the correspondent node after their initial communication. The home agent provides the care-of address of the mobile node to the correspondent node so that they can communicate directly afterwards. This will improve the performance when they need to send multiple messages. However, it does not help when the correspondent node wants to send a single packet to the mobile node, which is usually the case in the search process of structured peer to peer systems. The biggest problem with using mobile IP is *deployment*. Up to now, mobile IP has not been widely deployed. Therefore, it is necessary for us to explore other approaches to support mobility for peer to peer systems.

III. RUAM-BASED MOBILITY SUPPORT

A. Assumption and Background

We treat the moving and the rejoining of mobile nodes differently from normal leaving and joining. We call the action of a mobile node starting moving as *mobile-leave*, and a mobile node rejoining as *mobile-join*. When a node initially joins the system, it acquires an ID, which can be obtained by hashing the initial IP address, the host name, or the private key of the node. This unique ID will remain with the mobile node regardless of how many times it mobile-leaves or mobile-joins the system.

RUAM is based on Chord [4] and extends it to support mobile nodes. Chord organizes the system topology as a ring. Let m be the number of bits in the identifier space, then a total number of 2^m identifiers are available in a system. They are used for both node IDs and object IDs. Key k of an object is stored at the first node whose ID is equal to or greater than k . This node is called the successor of k . If the identifiers of nodes are represented as numbers from 0 to $2^m - 1$ clockwise on a circle, then the successor of k is the first node clockwise from k . The predecessor of node n is the first node counterclockwise from n . Each node maintains a routing table (also called finger table) with m entries. The i^{th} entry in the finger table of n is the successor of the identifier $(n + 2^{i-1}) \bmod 2^m$, where $1 \leq i \leq m$. When a node receives a query, it sends a

reply if it stores the requested key, or it will route the query to a node whose ID is closer to the requested key. In a N -node network, if each node maintains $O(\log N)$ entries in the routing table, a lookup can be resolved in $O(\log N)$ hops with high probability [4].

B. Identifying related nodes

Assume that the current Chord ring is in an ideal state[8], that is, each finger table entry is correctly pointing to a valid node. Suppose that p is a mobile node and q is its predecessor. Recall that the related nodes of p are the nodes having at least a finger pointer pointing to p . To locate p 's related nodes, we identify m ranges in the identifier space, each of which is represented as R_i , where $1 \leq i \leq m$. The smallest and largest identifiers of range R_i (denoted as $R_i.first$ and $R_i.last$, respectively) can be determined as follows.

$$R_i.first = (q - 2^{i-1} + 1) \bmod 2^m$$

and

$$R_i.last = (p - 2^{i-1}) \bmod 2^m$$

From the peering relationships, we know that range R_i contains the nodes whose i^{th} fingers are pointing to p . For example, figure 1 shows R_m and R_{m-1} of the mobile node p . r is in R_m , and s is in R_{m-1} . Since both r and s are the related nodes of p , they have at least a finger pointing to p . More specifically, the m^{th} and $(m-1)^{th}$ finger of r and s are pointing to p , respectively.

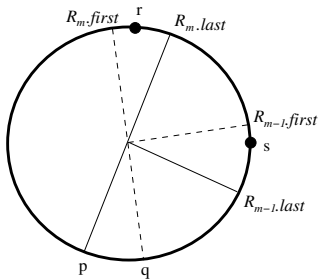


Fig. 1. An example of related nodes

C. Mobile-leaving

Having identified m ranges, a node is able to actively update each node in the ranges from R_m to R_1 . The key question is to find the first node in each range. We propose a method based on using the predecessor of a mobile node to help with the update. When a node mobile-leaves the system, it sends an updating message to its predecessor. In turn, the message is forwarded to the node pointed by the predecessor's longest finger. The updating procedure then starts from that node. For example, when p mobile-leaves, p sends an updating request to its predecessor, q . Notice that the longest finger of q is pointing to the first node which is either equal to or succeeding the identifier $(q + 2^{m-1}) \bmod 2^m$, and this identifier immediately precedes $R_m.first$. Thus, q 's longest finger

node is either immediately preceding range R_m or the first node in range R_m . By starting at this node, we guarantee that the message efficiently approaches R_m as well as no related nodes of p precede this node.

Algorithm 1 n.RUAM($p, q, i, p_succ, status, ts$)

```

1: if  $ts$  is the latest and  $i \geq 1$  then
2:   if  $n \in R_i$  then
3:     update  $n$ 's finger table
4:     if  $n.successor \in R_i$  then
5:        $n.successor.RUAM(p, q, i, p\_succ, status, ts)$ 
6:     else if  $n.successor \notin R_i$  then
7:        $next = n.closest(R_{i-1}.first)$ 
8:        $next.RUAM(p, q, i-1, p\_succ, status, ts)$ 
9:     end if
10:  else if  $n \notin R_i$  then
11:    if  $n.successor$  is before  $R_i$  then
12:       $next = n.closest(R_i.first)$ 
13:       $next.RUAM(p, q, i, p\_succ, status, ts)$ 
14:    else if  $n.successor$  is in  $R_i$  then
15:       $n.successor.RUAM(p, q, i, p\_succ, status, ts)$ 
16:    else if  $n.successor$  is after  $R_i$  then
17:       $n.RUAM(p, q, i-1, p\_succ, status, ts)$ 
18:    end if
19:  end if
20: end if

```

Algorithm 2 n.closest(id)

```

1: for  $i = m$  downto 1 do
2:   if  $finger[i].node \in (n, id)$  and it is alive, not moving
3:     then
4:        $pred = finger[i].node$ 
5:       return  $pred$ 
6:   end if
7: end for
8: return  $n$ 

```

The pseudocode of the updating procedure is given in Algorithm 1, in which i indicates the range that the algorithm is updating, and ts denotes timestamp. Here we use an example in Figure 2 to illustrate how the algorithm works. In Figure 2, node r and t are in R_i . They are the first and the last node in R_i , respectively. Node s is in R_{i-1} . There are several nodes in between R_i and R_{i-1} . Among them, only node u is in r 's routing table. Node v is the predecessor of $R_{i-1}.first$.

When a node in range R_i receives the updating message, after updating itself, it sends the message to its successor if the successor is also in range R_i (line 3-5). This step guarantees all nodes in R_i can see the updating message. In the example, when r receives an updating message, it updates itself and checks its successor. Node r passes the message to t since t is also in R_i . Similarly, t updates itself and checks its successor. However, t 's successor is out of range R_i .

For a node which is in R_i but its successor is out of R_i , it updates itself and then finds the closest node

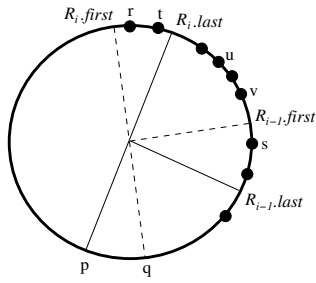


Fig. 2. An example of RUAM

to $R_{i-1}.first$ from its finger table and forwards the message to that node (line 6-9). The message contains the next updating range, R_{i-1} , to indicate all nodes in R_i have been updated. Algorithm 2 shows the procedure for finding the closest node to a given identifier from a local finger table. Basically, when passed with an id , the node examines fingers from the longest to shortest one, and it returns the first available finger node preceding the given id . Continuing with our example, from t 's finger table, t learns that u is the closest node to $R_{i-1}.first$. Thus t forwards the message to u . Also, t specifies R_{i-1} as the next range in the message.

If a node receives the message and it is not the predecessor of the specified range, it routes the message to the node closest to $R_{i-1}.first$ (line 11-13). In this way, the message is routed through a sequence of finger nodes until the predecessor of $R_{i-1}.first$ is reached. Using finger nodes to relay messages efficiently delivers the updating message from one range to another.

Once the predecessor of $R_{i-1}.first$ receives the message, it forwards the message to its successor to update nodes in R_{i-1} (line 14-15). In the example, when v receives the message, it sends the message to s through its successor pointer to update nodes in R_{i-1} .

However, if the predecessor of $R_{i-1}.first$ cannot find a node in R_{i-1} (*i.e.*, its successor succeeds $R_{i-1}.last$), the predecessor thus knows no nodes exist in R_{i-1} . As a result, it starts looking for the closest node to $R_{i-2}.first$ (line 16-17). The updating process terminates when the message reaches the last node of R_1 .

D. Mobile-joining

When a node mobile-joins the system, it updates its related nodes in the same way as when it mobile-leaves. However, the node itself may also contain stale fingers since it cannot receive updating messages due to its disconnection. To update them, the node asks for a finger table from its successor and appropriately updates entries. Therefore, the node avoids using the joining procedure to re-attach itself in the system.

E. Searching

The searching algorithm provided by a stationary P2P system should be changed accordingly because of the existence of mobile nodes. While preserving the existing searching algorithm, we add features to it so as

to incorporate mobile nodes into the system as well as to improve searching performance. First, if intermediate nodes are available (*i.e.* not moving), a query is routed in the same way as it is in a stationary P2P system. Once an intermediate node is disconnected due to moving, a node selects an appropriate node, which is currently available and the closest to the queried key in its finger table, as the next hop to forward the query. Second, when the query's root node is found unavailable, its predecessor returns the root node's information to the query source. Last, each node in RUAM maintains a successor list which contains several successors. If a query can not be forwarded through a node's fingers, it will be sent through nodes in the successor list.

F. Discussions

To implement the RUAM, we augment three fields to each entry of a finger table, called STATUS, ALT_SUCC, and timestamp. The STATUS field can be set to one of the three values, NORMAL, MOBILE, and DEAD. NORMAL means the node is alive and not moving. MOBILE implies the node is alive but temporarily disconnected from the system. DEAD indicates the node can not be contacted.

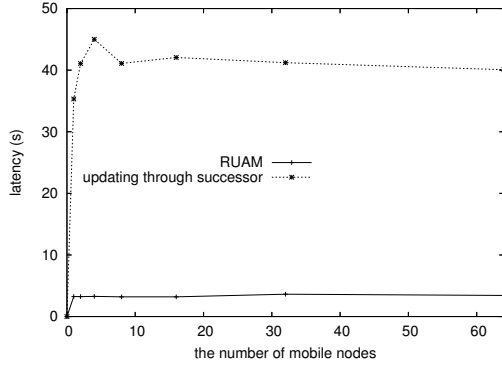
The ALT_SUCC is an alternative finger pointer when the entry's status is MOBILE. For instance, when p mobile-leaves, p encloses its successor in the updating message, thus, p 's related nodes not only update the status to MOBILE, but also set ALT_SUCC to p 's successor. Therefore, p 's related node can still route queries efficiently even though p is moving.

The timestamp field is used to indicate the freshness of an entry. Since congestion in network layer is unavoidable, there is no guarantee of the order in which a node receives messages. In case of when node p mobile-leaves and soon it returns, p 's related node may first receive the message which is sent when p mobile-joins, and later, the related node receives the message sent when p mobile-leaves. Without the help of timestamp, p 's related node may falsely set p 's status to MOBILE even though p has returned. Therefore, a related node only accept messages with the latest timestamp. The value of timestamp is set by a node when it mobile-leaves or mobile-joins.

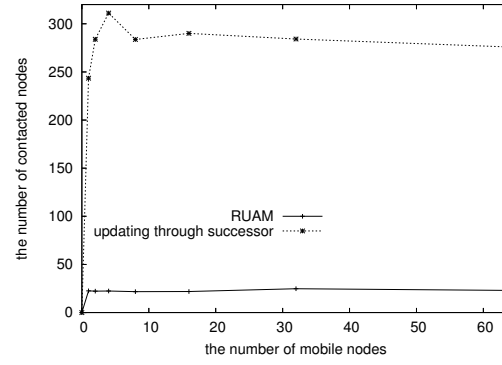
IV. SIMULATION RESULTS

We use *p2psim* [15] to construct our simulation system. Its event generator is modified so that it is able to generate mobile events in addition to churn events. *p2psim* uses three periodic procedures, *basictimer*, *succlsttimer*, and *finger timer*, to maintain finger tables and successor lists. Their default values are 100 seconds.

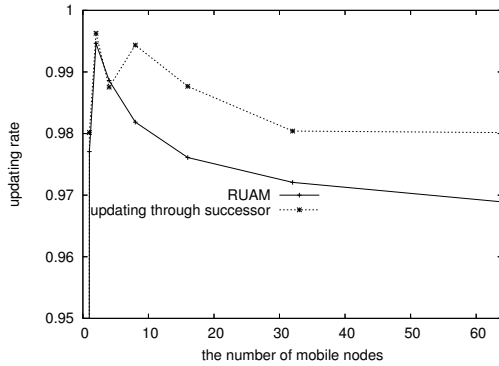
Our simulator uses *GT-ITM*[16] to produce a topology with 1600 nodes. Among the 1600 nodes, 600 nodes are randomly chosen to build a Chord system. In our simulation, the network latency is set to 10 ms. The lookup interval of each node follows exponential distribution with a mean value of 10 second. Each node maintains three nodes as its successors.



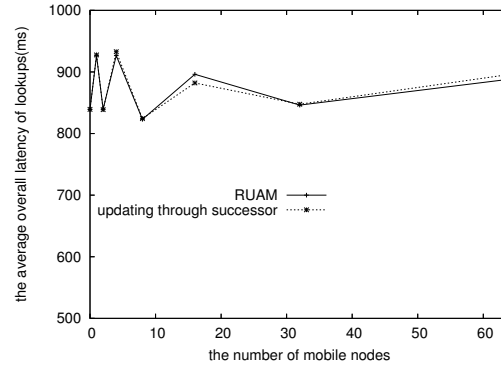
(a) the average updating latency



(b) the average number of contacted nodes



(c) the average updating rate



(d) the average latency of overall lookups

Fig. 3. The performances of RUAM and updating through successor

Our mobile model is defined by three parameters, *start_moving_time*, *mobile_time*, and *mobile_life*. We use *start_moving_time* to indicate how soon a node will start to move after it initially joins the system. *mobile_time* denotes the time interval between a mobile-joining and the next mobile-leaving. It shows how long a node stays in the system after its previous mobile-joining. *mobile_life* indicates how long a node is offline after each mobile-leaving. They are exponentially distributed with mean values of 600, 300, and 300 seconds, respectively.

The following metrics are used to evaluate the performance:

- *Updating rate*. For an updating message, updating rate is the ratio of the number of the nodes which receive the message and update their finger tables to the total number of nodes which should receive the message.
- *Updating latency*. It is the time the update algorithm gets the nodes updated.
- *Number of the contacted nodes*. This metric indicates the number of contacted nodes to finish a updating process. This metric also indicates the overhead of the updating process.
- *Relative success rate of lookups*. This metric is a ratio of the success rate of the lookups in a system with

mobile nodes to the rate of a system without mobile nodes.

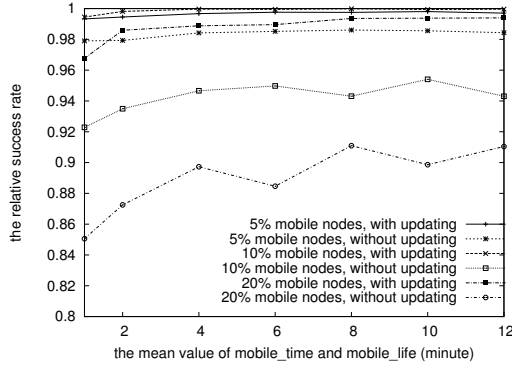
- *The latency of a lookup*. This metric indicates how soon a lookup terminates.

A. Updating performance

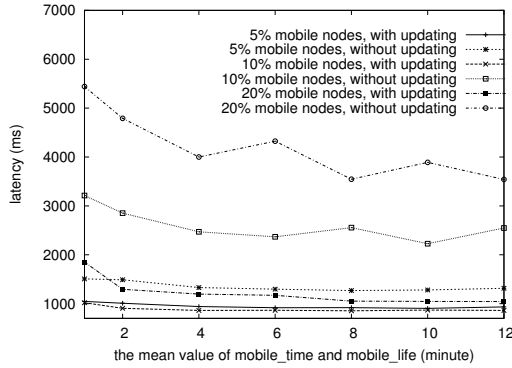
We compare RUAM with a naive updating algorithm, called *updating through successor* in the following sections. Let p and q still denote a mobile node and its predecessor. Similar to RUAM, the naive algorithm starts the updating process at q 's longest finger node. Afterwards, the updating message is forwarded though successor pointers from one node to another till q receives the message. If an intermediate node is a related node of p , it updates itself and forwards the message to its successor. Otherwise, it simply forwards the message.

We design two groups of tests to evaluate our algorithm. One is for updating performance and the other is for lookup performance. Figure 3(a), 3(b), and 3(c) compare the updating performance. Figure 3(d) compares the lookup efficiency.

In Figure 3(a) and 3(b), we see that RUAM significantly reduces both the updating latency and the number of contacted nodes by a factor of 10. These improvements indicate that RUAM efficiently reduces messages forwarded



(a) the relative success rate of overall lookups



(b) the average latency of overall lookups

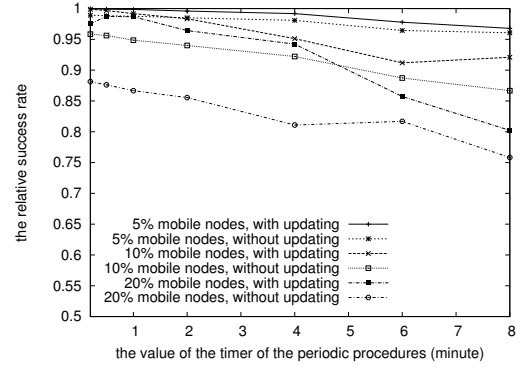
Fig. 4. The performances of lookups with updating and without updating when the rate of mobile events changes

between unrelated nodes. Figure 3(c) shows that RUAM performs slightly worse than the naive algorithm in updating rate. The reason is that the system is not ideal as we assumed in Section III. There exist inconsistent entries in finger tables. Therefore, even the naive algorithm cannot achieve 100% updating rate. However, the updating rate of RUAM is only 1% worse than the naive algorithm. Figure 3(d) reveals a comparable lookup performance of both algorithms. From this set of figures, we know that RUAM can achieve comparable lookup efficiency while greatly reduce the update latency and the number of messages exchanged.

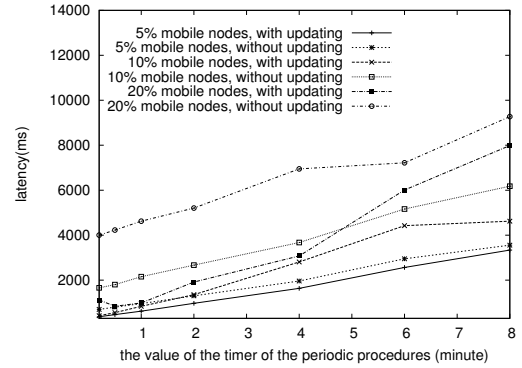
B. Lookup performance

In this section, we investigate the lookup performance of using RUAM and of using no updating algorithm at all. We refer to them as the performance of *lookups with updating* and *lookups without updating*, respectively. Without using any updating algorithm, a node leaves a system silently and does not send any messages. We conduct the simulations with 5%, 10%, 20% mobile nodes.

First, we consider above settings in different mobile models by changing the mean values of *mobile_time* and



(a) the relative success rate of overall lookups



(b) the average latency of overall lookups

Fig. 5. The performances of lookups with updating and without updating when the intervals of the periodic procedures change

mobile_life. In this group of experiments, we let both mean values equal to each other. Figure 4 shows that the performance of *lookups with updating* is better. Moreover, in both sub-figures, for a given *mobile_time(mobile_life)*, the performance degrades as the percentage of mobile nodes increases. For example, in Figure 4(b), when the *mobile_time(mobile_life)* is 6 seconds, the latency in *lookup without updating* is almost doubled if the percentage of mobile node changes from 10% to 20%. However, the performance is only slightly degraded in *lookups with updating*. This indicates RUAM efficiently updates finger tables. We notice that for a given percentage of mobile nodes, the performance also improves as the *mobile_time(mobile_life)* increases. This may be caused by the mobile churn. The smaller the *mobile_time(mobile_life)*, the bigger the churn. As a result, more finger pointers become inconsistent.

Second, we simulate the lookup performance by using different interval values of periodic procedures. In Figure 4, we see that all curves change smoothly as the *mobile_time(mobile_life)* increases, especially the curves for *lookups with updating*. Therefore, we can derive that the number of mobile nodes affects the performance more than the two mean values do. Thus, in this part, we do not change the mobile model, instead change the number

of mobile nodes. The mean values of *mobile_time* and *mobile_life* are set to 600 seconds.

In Figure 5, *lookups with updating* still outperforms *lookups without updating* for a given percentage of mobile nodes. Especially when the intervals are less than 1 minute, we see from curves of *lookups with updating* that the latency is less than 1000ms, and the relative success rates is greater than 97%. For given intervals of the periodic procedures, when the percentage of the mobile nodes is 5%, the performance of *lookups with updating* differs very slightly from that of *lookups without updating*. Hence, with a small percentage of mobile nodes, the performance depends on the value of intervals. When the intervals of the periodic procedures are within the range from one to six minutes, the more the mobile nodes, the more benefit gain can be achieved by *lookups with updating*. For example, in Figure 5(b), when the interval values are four minutes, the benefit gained in latency by using *lookup with updating* is about 400ms less than the latency of *lookup without updating* if 5% nodes are mobile nodes. However, the benefit gain is about 4500ms less if 20% nodes are mobile nodes. When the intervals are greater than 6 minutes, no matter whether the percentage of the mobile nodes is 20% or 10%, the benefit gain between them is small. This phenomenon indicates that the efficiency of updating becomes limited when the intervals are large. Since the periodic procedures are not activated frequently, there might exist some long-time inconsistent fingers and successors. However, RUAM cannot make them consistent because it is only used by the mobile node to update its information in its related nodes. Therefore, the intervals of the periodic procedures cannot be set too large in order to maintain the consistency of the fingers and successors in the mobile computing environment.

Based on the simulation results, we conclude that when the percentage of mobile nodes is small, RUAM improves the lookup performance slightly. In contrast, when the percentage of mobile nodes is large, RUAM improves the lookup performance significantly if the intervals of the periodic procedures are in a reasonable range. In addition, if the intervals of the periodic procedures are very large, the efficiency of RUAM will be decreased as the intervals increase.

V. CONCLUSION

This paper has presented RUAM to support node mobility in structured P2P systems. Through simulations, we compare RUAM with a naive updating algorithm in their updating and the resulted lookup performance. The results show that RUAM significantly reduces the updating latency while both algorithms have similar effects on lookups. In addition, we investigate the lookup performance in different mobile models, and how the lookup performance is related to the intervals of the periodic procedures. Two conclusions can be drawn. First, RUAM outperforms *lookups without updating* in mobile computing environment. Second, if the intervals of the

periodic procedures are small, the larger the percentage of the mobile nodes, the more improvement can be achieved by RUAM.

VI. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under Grants CCR-0204304 and EIA-0101242, and a grant from the Kentucky Science and Engineering Foundation as per Grant Agreement #KSEF-148-502-05-139 with the Kentucky Science and Technology Corporation.

REFERENCES

- [1] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica. "Wide-Area Cooperative Storage with CFS," Proceedings of the 18th Symposium on Operating Systems Principles, Banff, Canada, October, 2001.
- [2] Antony Rowstron, Anne-Marie Kermarrec, Peter Druschel, Miguel Castro. "Scribe: The Design of a Large-Scale Event Notification Infrastructure," Proceedings of The Third International Workshop on Networked Group Communication (NGC), London, UK, November, 2001.
- [3] Angelos D. Keromytis, Vishal Misra, Dan Rubenstein. "SOS: Secure Overlay Services," Proceedings of ACM Sigcomm'02, Pittsburgh, PA, August, 2002.
- [4] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 17-32, February 2003.
- [5] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, John Kubiawicz. "Tapestry: A Resilient Global-scale Overlay for Service Deployment," IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1, January 2004.
- [6] A. Rowstron, P. Druschel. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November, 2001.
- [7] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker. "A Scalable Content-Addressable Network," ACM SIGCOMM Conf., San Diego, CA, September 2001.
- [8] David Liben-Nowell, Hari Balakrishnan, and David Karger. "Analysis of the Evolution of Peer-to-Peer Systems," ACM Conf. on Principles of Distributed Computing (PODC), Monterey, CA, July 2002.
- [9] Charles E. Perkins. "Mobile IP," Communication Magazine, May, 1997.
- [10] Shelley Q. Zhuang, Kevin Lai, Ion Stoica, Randy H. Katz, Scott Shenker. "Host Mobility using an Internet Indirection Infrastructure," Proceedings of the First International Conference on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys), May, 2003.
- [11] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, Sonesh Surana. "Internet Indirection Infrastructure," Proceedings of the SIGCOMM 2002, August, 2002.
- [12] Ben Y. Zhao, Ling Huang, Anthony D. Joseph and John D. Kubiawicz. "Rapid Mobility via Type Indirection," Proceedings of the Third International Workshop on Peer-to-Peer Systems (IPTPS'04) San Diego, CA, February, 2004.
- [13] Hung-Chang Hsiao, Chung-Ta King. "Mobility Churn in DHTs," Proceedings of the First International Workshop on Mobility in Peer-to-Peer Systems (MPPS'05) in conjunction with the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), Columbus, OH, June, 2005.
- [14] Hung-Chang Hsiao, Chung-Ta King. "Bristle: A mobile Structured Peer to Peer Architecture," Proceedings of the International Parallel and Distributed Processing Symposium. IEEE Computer Society, April 2003.
- [15] p2psim.<http://pdos.csail.mit.edu/p2psim/>.
- [16] Ellen W. Zegura, Ken Calvert and S. Bhattacharjee. "How to Model an Internetwork," Proceedings of IEEE Infocom '96, San Francisco, CA.