Mobility Churn in DHTs

Hung-Chang Hsiao[†] and Chung-Ta King[‡*]
[†]Computer and Communication Research Center
[‡]Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan 300
{hchsiao, king}@cs.nthu.edu.tw

Abstract

Most peer-to-peer (P2P) overlays based on distributed hash tables (DHTs) focus on stationary Internet hosts. However, when nodes in the last-mile wireless extension are allowed to join the overlay, we will face immediately the problem of peer mobility. That is, when a peer moves to a new location in the network, all the state information regarding the moving peer will become stale, resulting in creating mobility churn to the system in addition to ordinary churn due to peers joining, departure and failure. The paper extensively investigates the performance of a DHT that is operated under mobility churn. The DHT relies on rudimentary failure detection/recovery mechanisms and peer dynamic joining/departure algorithms to handle peer mobility. We show that when compared with an ideal design, rudimentary failure handling mechanisms and peer dynamics processing algorithms often take unhelpful maintenance bandwidth. The rudimentary implementation has fair performance results when ordinary churn rate and/or route request rate is tremendously high.

1 Introduction

A *peer-to-peer* (P2P) overlay aims to provide a universal utility space that is distributed among the participating end computers (peers) across the Internet. An exciting strategy of organizing peers and data items in a P2P overlay is the use of *distributed hash table* (DHT) [1–5], which represents peers and data items with unique hash keys to control the topology and data placement in the overlay network. Since DHT-based overlays (abbreviated as DHTs) rely on a logic structure (e.g., a ring in Chord [4] and a torus in CAN [2]) to organize the interconnection of the peers, they are also called *structured* P2P overlays.

A common strategy of naming and routing in DHTs is to store a data item with a hash key k in a peer node whose hash key (or ID) is the numerically closest to k. To fetch the data item, the request message is routed through the intermediate peers whose hash keys are closer and closer to k. This requires that each peer node keeps some state information for routing. The state information may appear as a list of entries, each is referred to as a *state* in this paper. A state associates, among other things, the hash key of a known peer and its network attachment point, e.g., the IP address and port number. A state thus gives the location information of a peer node in the network, which allows the local node to communicate with that node directly. When a node receives a message, it forwards the message to the next peer according to the states that it maintains. Obviously, states in a DHT-based P2P overlay are distributed, and a robust and scalable DHT relies on maintaining these states.

Previously proposed DHTs, including CAN [2], Chord [4], Pastry [3], Tapestry [5], Tornado [1], etc., assume that the participating peers are stationary. This is because DHTs are originally proposed for Internet-based wide-area applications. However, a growing number of devices are now connected to the Internet through wireless links. Wireless devices may want to join a DHT-based overlay and access its services or resources. For example, a wireless device may like to join a multicast group (e.g., a group of players in a multiplayer game), where all members are DHT nodes, in order to receive messages addressed to that group. The problem is that nodes may move and we will have the *peer mobility* problem. The state regarding a moving node may be distributed throughout the P2P overlay and will become stale. We will refer to such a type of DHTs as *mobile DHTs*.

Typical DHTs suffer from dynamics, called the *ordinary churn*, due to peer joining, departure and failure. Unlike typical DHTs, mobile DHTs, however, experience another type of churns–*mobility churn*–due to peer mobility. Such an additional churn with ordinary one may further degrade routing efficiency and increase maintenance overheads. However, it would be possible to handle the mobility churn using naive failure detection and recovery mechanisms designed for handling ordinary churn to discover and repair failures. That is, stale states created due to peer movement can be detected using the failure detection mechanism and/or even be recovered with the recovery one. On the other hand, when a mobile peer changes its network location, it creates its new state

[*]C.T. King is supported in part by the Ministry of Education, Taiwan, under Grant NSC 93-2752-E-007-004-PAE.

to the system using the peer departure algorithm followed by the join.

To our best knowledge, most DHTs do not differentiate ordinary and mobility churns, which depend on rudimentary failure handling mechanisms and peer dynamic processing algorithms inherit in the DHTs to cope with the mobility churn. In this study, we intend to answer the fundamental problem, that is, whether the rudimentary solutions can efficiently handle the mobility churn. We concentrate on a mobile DHT that mixes stationary and mobile peers. To simplify the discussion, we do not consider the environment in the absence of an routing infrastructure as discussed in [6]. Similar to the previous study [7], we assume that a mobile peer can simply connect to the system via its nearby wireless access point; the peer receives a new network address by consulting a local DHCP server. We do not assume that mobile IP infrastructure is available since having a mobile IP infrastructure requires considerable deployment effort and thus cost. The mobile DHT thus requires handling peer movement itself.

Through extensive simulations, we conclude that a DHT with the rudimentary failure handling mechanisms and peer dynamic processing algorithms can have fair performance results (in terms of delay of object locating and routing) when the system is operated under a tremendously high ordinary churn and/or a very high route request rate. However, the rudimentary implementation often introduces helpless maintenance overheads to the system.

2 Problem Statement

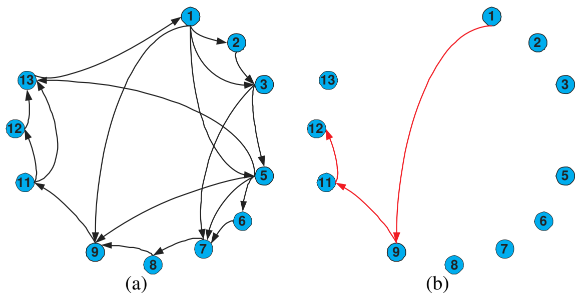


Figure 1. (a) An example of a DHT with 11 peers and (b) n_1 initiates a route towards n_{12}

Figure 1(a) shows a possible DHT configuration [1], where there can be 16 peers (denoted as $\mathcal{R}$) in the system at most. The node n_1 maintains four neighbors, namely n_2, n_3, n_5 and n_9. n_2, n_3, n_5 and n_9 are level-4, -3, -2 and -1 neighbors [2], respectively. To route a message from n_1 to n_{12}, n_1 can firstly forward the message to n_9 since n_9 is the neighbor of n_1, which has the hash ID closest to that of n_{12} (see Figure 1(b)). Upon receiving the message, n_9 performs the similar forwarding process by forwarding the message to n_{11} because n_9 only keeps n_{11} as its neighbor at that movement. The message finally reaches n_{12}, which is sent from n_{11}.

As mentioned, peer mobility creates churn to a mobile DHT. Assume that n_9 is moving and has not reattached to the network. In the meantime, n_1 wants to send a message to n_{12} and tries to forward the message to its level-1 neighbor, n_9. However, n_1 is unaware of that n_9 is moving and thus the forwarding fails. After a number of retries are done and no response is from n_9, n_1 seeks for another route by forwarding the message to its level-2 neighbor, n_5. n_5 performs the similar. Again, the forwarding fails until n_5 successfully relays the message to n_7. n_7 then relays the message to n_8, and finally the messages reaches n_{11} and the destination n_{12}. We note that in a ring-based DHT such as Chord, Pastry, Tapestry and Tornado, peers aggressively maintain the ring structure, resulting in that n_8 has n_{11} as its closest neighbor when n_9 has not reconnected to the network. n_8 thus can relay the message towards the destination.

A typical mechanism to implement the retry is to wait for a time period (call the *timeout period*) in order to receive an acknowledge message from the receiver. If the sender does not receive any response from the receiver after the timeout period elapses, the sender will perform another try for the forwarding. After a number of retries, the sender can then confirm that the receiver is unreachable. Adopting the timeout and retry mechanisms can avoid message lose due to network congestion or transient outage.

Therefore, if each peer needs to take four retries in order to determine that a neighbor is unreachable, then performing a route in the above example will totally take 15 messages, and 7-hop delay (in which 2-hop delay is contributed by timeouts [3]) to reach the destination.

Apparently, before n_1 issues the route, if n_1 can detect that n_9 is unreachable in time and replaces n_9 with another peer (e.g., n_8) as its level-1 neighbor, n_9 only needs to take three messages and hops to perform the route. This can considerably reduce the messages required up-to 12; shorten the hop count up-to 4.

Afterwards, n_9 returns to the system with a new network attached point. Subsequently, n_1 issues a message towards n_{12} [4], which will still forward its message to n_5 if n_1 has

[1]In this paper, our discussion is in the context of Tornado [1]. However, we believe that the results discussed in this paper will not qualitatively varied when considering other DHTs such as Chord, Pastry and Tapestry.

[2]The way Tornado chooses the neighbors for each node is similar to that of Chord.

[3]We assume that timeouts required to identify a node that is unreachable takes one-hop delay.

[4]We assume that n_1 does not cache the address of n_{12} after their previous communication. This is reasonable since n_1 may only know a limited number of peers in the system due to the scalability concern. Before n_1

not performed neighbor discovery for seeking a new level-1 neighbor (e.g., n_8). If n_5 is also unaware that n_9 has returned, then the message will visit n_5, n_7, n_8, n_9, n_{11} and n_{12}. Since n_1 and n_5 have learned that n_9 was unreachable previously, they do not need to take additional overheads to perform their retries. This leads to the message to take six messages and hops for reaching its destination.

Again, if n_1 recovers n_9 in time before it issues the message towards n_{12}, the message only needs to take three messages and hops to reach its destination.

It is possible that n_9 moves to a new location and soon reattaches to the system, n_1 does not know such a transient movement. Therefore although n_9 is present to the system, n_1 still requires to perform retires in order to forward a message towards n_{12}. This is because the network address corresponding to n_9's state stored in n_1 is not up-to-date. Consequently, n_1 requires forwarding the message to n_5. If n_5 is also unaware of n_9's transient movement, n_5 also requires to take some retries to ensure that the previous address allocated to n_9 is unreachable, and seeks the help from n_7 to relay the message. The total number of messages and hops required for the route are 16 and 8 (2 out of 8 are due to timeouts), respectively.

We define the *ideal routes* and *maintenance bandwidth* as follows.

Definition 1: An *ideal route* from a peer n_i to another peer n_j ($n_i \Rightarrow n_j$) for a message m is if any peer n_k on the routing path does not experience any timeout (i.e., $\varrho(n_i \Rightarrow n_j) = 0$) when it forwards m to a peer towards n_j.

Definition 2: Given an overlay $G^{t_1}(V_1, E_1)$ at time t_1, it evolves into $G^{t_2}(V_2, E_2)$ at time t_2, where $t_1 < t_2$; V_1 (E_1) and V_2 (E_2) are the sets of nodes (overlay links) in $G^{t_1}(V_1, E_1)$ and $G^{t_2}(V_2, E_2)$, respectively. The *maintenance bandwidth* of a node $n_i \in V_2$ is $\beta_i = \frac{\tau_i \times b}{t_2 - t_1}$, where τ_i is the total number of messages required by $G^{t_1}(V_1, E_1)$ to evolve into $G^{t_2}(V_2, E_2)$ and b is the number of bytes per message. The average maintenance bandwidth $\overline{\beta}$ from t_1 to t_2 is $\frac{\sum_{n_i \in V_2} \beta_i}{|V_2|}$.

As we will discuss later, the bandwidth required by a peer to maintain a DHT depends on several parameters (e.g., the failure detection and recovery periods). In this study, we assume that each peer has a constant network bandwidth allocated to maintain a DHT. This is desirable since a peer may like to control its network resources contributed to the system. Thus, we expect an ideal design of a mobile DHT is as follow.

Definition 3: For any node n_i in the overlay can take the maintenance bandwidth budget β_i, an *ideal mobile DHT* $G^{t_2}(V_2, E_2)$ evolved from $G^{t_1}(V_1, E_1)$ is that a route request issued at t_2, after the last route issued at t_1, from $n_i \in V_2$ towards any node $n_j \in V_2 - \{n_i\}$ is an ideal route, i.e., $\varrho(n_i \Rightarrow n_j) = 0$.

talks to n_{12} again, it may have communicated with many other peers and thus have discarded the address of n_{12}.

For simplifying the discussion, in this study we assume that the bandwidth budget allocated by each peer is identical.

Before devising any mechanism or algorithm to have an ideal mobile DHT, we are interested to know *under the maintenance bandwidth budget given to each peer, whether rudimentary failure detection/recovery mechanisms and peer dynamic processing algorithms can enable peers in a mobile DHT to experience less timeouts when relaying a message*. We will discuss a rudimentary implementation later.

3 Protocol Design Consideration for Mobile DHTs

Most DHTs include peer joining/departure algorithm and failure detection/recovery mechanisms to handle dynamics. We discuss how mobility churn affects the design. Before that, we formally define the following terminologies.

Definition 4: Any mechanism that can identify and remove a faculty state s from $state[n_i]$ for any node $n_i \in V$ is called the *failure detection mechanism*, where $state[n_i]$ denotes the states hosted by n_i and V includes all nodes in the overlay.

Definition 5: Any mechanism that can include a state s into $state[n_i]$ for any node $n_i \in V$ is called the *failure recovery mechanism*.

Clearly, all states in the system will be $\bigcup_{n_i \in V} state[n_i]$. DHTs such Chord, Pastry, Tapestry and Tornado has $|state[n_i]| = O(\log |V|)$ for any node $n_i \in V$.

Definition 6: Any algorithm that designates which of states in $\bigcup_{n_i \in V} state[n_i]$ should be included/excluded to/from $state[n_i]$ for any node $n_i \in V$ is called the *node joining/departure algorithm*.

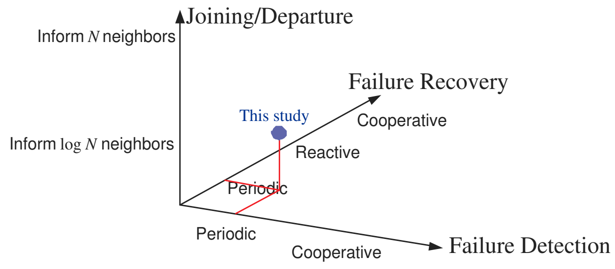


Figure 2. The design space regarding the peer dynamic handling algorithms, and the failure detection and recovery mechanisms

Figure 2 depicts the design space regarding the peer joining/departure algorithm and the failure detection/recovery mechanism. We note that the states hosted by n_i may not be up-to-date as well as those in $\bigcup_{n_i \in V} state[n_i]$. Therefore, a well-designed peer joining/departure algorithm needs to avoid including/excluding stale/fresh states for any node n_i. On the other hand, if a failure detection/recovery mechanism can remove/introduce stale/fresh states from/to $state[n_i]$ as much as possible, then the joining and departure algorithms can simply concentrate on how to optimize the performance metrics such as the routing delay and the network bandwidth for any two given nodes by appropriately selecting states for each node n_i.

For the peer joining/departure algorithm, as defined above a node hosts a number of designated states. In the study, our implementation designates states of those representing peers whose IDs closest to $x + \frac{\mathcal{R}}{2^i} \bmod \mathcal{R}$ to a node x (where $i = 1, 2, 3, \cdots, \log N$), resulting in that the total number of states a peer maintains is $\log N$. However, one may design a peer joining algorithm to have a node host more than $\log N$ states. In [8], a node maintains N states, for example.

For the failure detection mechanism, the commonly implemented in most DHTs is the periodic mechanism [9]. In addition to the periodic mechanism we can enable each node to cooperatively inform about the detected transient movement of a mobile node to nodes that have the mobile node as their neighbor. Consequently, these nodes do not need to consume additional network bandwidth to detect that failure due to timeout and retries. However, efficiently disseminating a detected failure depends on the number of states per node maintained in the system. For example, we may like to have an efficient broadcast communication to notify a detected faculty state in the case of that each node keeps the all states of the system. In contrast, if a node only keeps $\log N$ states, implementing a scalable multicast communication for state dissemination may be appropriate.

The failure recovery mechanisms can be periodic, reactive, cooperative, etc. The reactive mechanism is triggered due to detected failures. For example, a node x can soon invoke the discovery for its $\log N$ neighbors with IDs closest to $x + \frac{\mathcal{R}}{2^i} \bmod \mathcal{R}$ upon detecting a failure. It is also possible to mix the cooperative failure detection and recovery mechanisms. That is, the failure discovery is triggered if a local node receives the notification from a non-local node that detects a failure. Finally, similar to the cooperative failure detection, nodes with the cooperative failure recovery cooperatively help recover states and advertise these discovered states each other. This can be useful if a node x receives an updated location of a mobile node y from another node z that has resolved y's present location, x can help broadcast the mobile node's present address to interested peers.

Another possible design axis that is not shown in Figure 2 is to take advantage of DHT storage. When a node moves to a new location, it can publish its up-to-date address to that storage space. Any node that wants to address to a mobile node and does not have the updated location of the mobile node can consult that storage first in order to resolve the mobile node's current location.

Possibly, the design is to bear the network locality in mind. For example, to reduce maintenance traffic cross the Internet as much as possible, a node can have a higher probing rate to a geographically closer neighbor.

3.1 Rudimentary Implementation

Emulating Peer Movement with Joining and Departure. When a peer x starts to move, the peer invokes the peer departure algorithm. The peer departure algorithm informs the neighbors with the IDs closest to x to re-install their closest neighbors. That is, if x_1 and x_2 are the neighbors with the IDs closest to x and $x_1 < x < x_2$, x_1 (x_2) will have x_2 (x_1) as its closest neighbor.

In this study, we assume that x knows it will have a transient leave from the system. Thus, x can simply inform its closest neighbors to reset up their closest neighbors, properly. If x is unaware of its leave, the system still works since a node can simply keep a number of closest neighbors. If x fails to inform its closest neighbors about its transient leave, its neighbors can finally detect that x is unreachable using periodic ping messages as discussed later.

When x reattaches to the system with a new network address, the node invokes the node joining algorithm to join the DHT. For example, in Chord, when a node x joins the system, it issues $\log N$ routes towards nodes whose ID are closest to $x + \frac{\mathcal{R}}{2^i} \bmod \mathcal{R}$, where $i = 1, 2, 3, \cdots, \log N$. These nodes then return their IDs to x. x can thus maintain states of these nodes locally. Apparently, such an algorithm selects particular states to be hosted in each node.

We note that it is possible to have x rejoin the system by issuing a single route in order to discover a closest neighbor in the present network so that x can be immediately inserted into the ring. After, x simply relies on the state recovery mechanism to discover other neighbors (see Section 3.1). In this study, we implement such a variant algorithm.

State Detection and Recovery. The periodic failure detection mechanism requires each node to periodically ping each of its closest and non-closest neighbors every t'_d and t_d seconds, respectively. When a failure is discovered after consecutively issuing a number of consecutive pings, the node simply marks the state of the faculty node as invalid.

For recovering stale states, the periodic failure recovery mechanism is to periodically discover fresh nodes from the system every t_r seconds. Similar to the peer joining algorithm in Chord, in this study we implement that each node periodically discovers a level-i neighbor for a node x by issuing an ordinary route message towards a node with ID that is closest to $x + \frac{\mathcal{R}}{2^i} \bmod \mathcal{R}$. x discovers its neighbors in a round-robin fashion (i.e., $i = 1, 2, 3, \cdots, \log N, 1, 2, 3, \cdots$). Each route takes $O(\log N)$ hops and thus visits $O(\log N)$ nodes. In this paper, we implement the iterative approach. That is, each visited node replies x its ID and one of its neighbors that can help relay x's message. In the meantime, each visited node examines whether or not x can be its neighbor. x then forwards the message to that neighbor. Consequently, such an iteractive routing allows x to discover $O(\log N)^2$ states from the network every $\log N$ discovery messages.

However, since each peer maintains a limited number of states, a peer may require to replace a state when discovering a new one, or simply to discard the discovered state. We implement the *least-recently-used* (LRU) policy. The policy replaces a state that is not touched recently. A state is touched if a peer sends a ping message to the peer corresponding to that state and then receives a pong from that probed peer. The LRU policy allows a peer to keep states as fresh as possible.

4 Performance Evaluation and Results

Due to space limitation, the details of experimental setting can be found in [10]. The performance metrics we measured are as follows:

- *Percentage of hop count due to timeouts* (PHT): The metric is defined as the percentage of the number of hops induced due to timeouts.
- *Percentage of maintenance bandwidth due to timeouts* (PBT): The PBT measured is the average maintenance bandwidth created due to timeouts. We refer to the message format in Gnutella. The message simulated has 91 bytes in size, which comprises of a 54-byte header for MAC, IP and TCP, and a 37-byte header (including the ID filed and the message type)[5] required for a DHT packet.

We compare the rudimentary solutions with a $likely\ ideal$ implementation (denoted as the IDEAL in the remaining paper). The likely ideal implementation is to let each mobile node invalidate "all" its states stored in the system before the mobile node begins to move. To remove all states regarding the mobile node, we freeze the simulator, and then collect and remove the states associated with the mobile node. After removing all its states (removing a state takes one message), the simulator is resumed. Although in the real world it is unlikely to freeze the system and to perform invalidation, it is possible to have a solution that can approximate the ideal one. If the rudimentary solutions have the similar performance results with those reported by the ideal one, then this implies that the rudimentary solutions are ready for efficiently coping with the mobility churn. Therefore, the likely ideal implementation helps identify that whether or not the rudimentary solutions is efficient.

We note that in the likely ideal implementation, a route may still experience timeouts due to the ordinary churn.

[5]See the PING and PONG messages defined by Gnutella.

Finally, the maintenance bandwidth budget allocated to each node in the rudimentary and likely ideal designs is set to 110 bytes/second (in most cases in this study). This is because such a demanded bandwidth allows the likely ideal implementation creating no additional maintenance bandwidth due to timeouts. In the rudimentary design, to control the maintenance bandwidth we simulate as follows. Each node monitors its maintenance bandwidth every $t_w = 10$ seconds. If a node generates exceeding maintenance traffic before t_w expires, then the node ceases its failure detection and recovery processes. After t_w elapses, the node resumes the failure handling processes and goes to the next t_w period.

4.1 Results

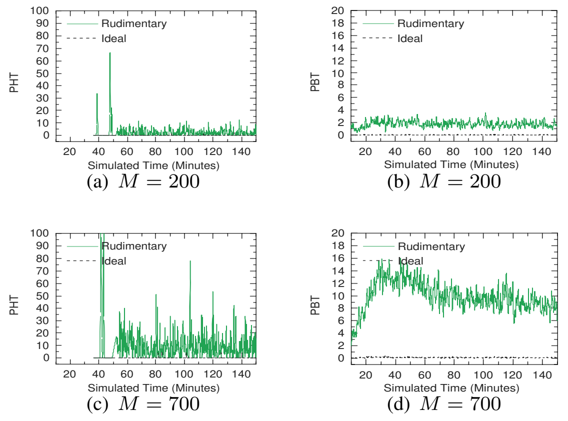
Effects of Different Numbers of Mobile Peers. We vary the number of mobile nodes, M, to 200, 500 and 700. The results for the numbers, $M = 200$ and $M = 700$, of mobile nodes are shown in Figure 3, and $M = 500$ in Figures 4(c) and (d). Clearly, when M is small (e.g., $M = 200$), a mobile DHT with the rudimentary failure detection/recovery and peer joining/departure algorithms experiences less timeouts and the route delay is thus not considerably lengthened. Also, the maintenance bandwidth includes less probing traffic required for detecting and recovering the stale states of mobile nodes. However, when M becomes large (e.g., $M = 500$ and $M = 700$), PHT and PBT can be markedly increased. When compared with IDEAL for large M's, the rudimentary solutions cannot perform well. That is, the rudimentary implementation takes helpless maintenance traffic (due to timeouts) while increasing route delay.

We note that in this experiment, the simulated DHT does not suffer from the ordinary churn created by peer joining, departure and failure. That is, we let each simulated DHT node have infinite lifetime. This allows us to identify the delay and bandwidth induced due to movements of peers.

Effects of Different Failure Detection Periods. We vary the failure detection period, t_d, to 30 and 300 seconds. Intuitively, if t_d is small, a peer is likely to detect a stale state cached locally soon at the expense of substantial network bandwidth. In contrast, if t_d is large, a peer takes less maintenance bandwidth, resulting in lengthening the route delay because of timeouts experienced by routing to a unreachable mobile peer.

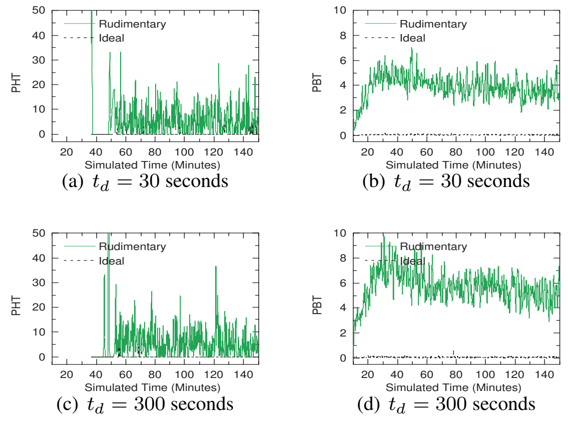
Figure 4 depicts the results. In the rudimentary implementation, PHT and PBT for $t_d = 30$ are slightly smaller than those for $t_d = 300$. Varying the failure detection period has a modest impact on PHT and PBT. Even with different t_d's, the rudimentary implementation still cannot perform as well as IDEAL.

Effects of Different Failure Recovery Periods. Figure 5 shows the results for different failure recovery periods ($t_r =$ 30 and 1800 seconds). Apparently, shortening the recovery period allows a peer to discover a fresh neighbor. This is



(a) $M = 200$ (b) $M = 200$ (c) $M = 700$ (d) $M = 700$

Figure 3. Effects of varying the number of mobile nodes



(a) $t_d = 30$ seconds (b) $t_d = 30$ seconds (c) $t_d = 300$ seconds (d) $t_d = 300$ seconds

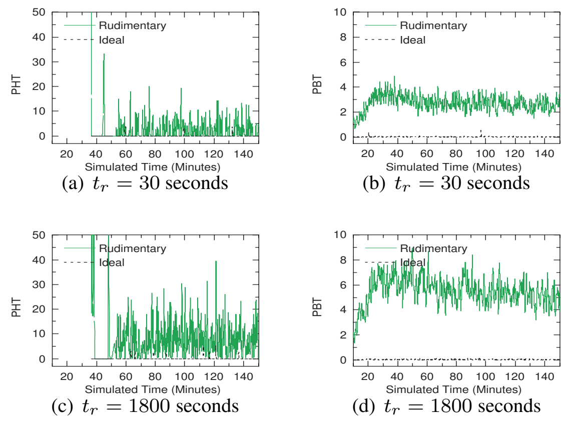
Figure 4. Effects of varying the detection period for non-closest neighbors

at the expense of introducing traffic to the system, though. Unlike results from varying the failure detection period, the recovery period can considerably affect the performance of the rudimentary implementation. If t_r is small (e.g., 30 seconds), the rudimentary implementation experiences less timeouts and generates less maintenance bandwidth when compared with IDEAL. However, when t_r is lengthened (e.g., 1800 seconds), PHT and PBT in the rudimentary implementation is clearly increased.

Effects of Different Peer Lifetime. In the previous experiments, we simulate each peer with infinite lifetime. Instead, the experimental results reported in Figure 6 are for each peer having the lifetime, t_l, with the means 5 and 50 minutes. The evaluation results present that If the system suffers from a high ordinary churn ($t_l = 5$), then the rudimentary implementation is comparable to IDEAL. This is because such a churn dominates the system dynamics. However, the rudimentary design still introduces more unhelpful maintenance bandwidth when compared with IDEAL.

Effects of Different Stationary Periods. In default, the peer mobility is medium. That is, a peer changes its network attachment point every 30 minutes (i.e., t_s) on average. We are also interested in the impacts of high and low peer mobility. Figure 7 illustrates the results for t_s equal to 5$\sim$10 and $50 \sim 480$ minutes. Clearly, in the rudimentary design, the high mobility churn ($t_s = 5 \sim 10$) can result in the longer delay and more maintenance bandwidth when compared with the low mobility churn ($t_s = 50 \sim 480$). Again, the rudimentary implementation cannot handle the mobility churn well no matter whether the mobility is high or low.

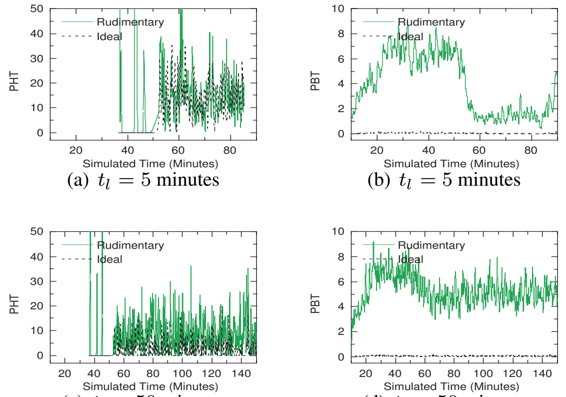
Effects of Different Route Request Rates. We finally investigate the effects of different route request rates (i.e., r).



(a) $t_r = 30$ seconds (b) $t_r = 30$ seconds (c) $t_r = 1800$ seconds (d) $t_r = 1800$ seconds

Figure 5. Effects of varying the recovery period

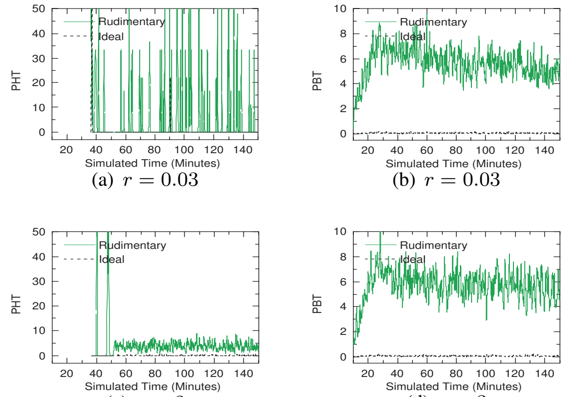
We simulate a peer that issues the number of routes from 0.03 to 3 per minute. As shown in Figure 8(c), if $r = 3$, the rudimentary implementation has the performance result similar to that of IDEAL. This is because given a mobility churn in which a node has a stationary period T on average, the larger r (i.e., the smaller time interval t between two consecutive route requests) is, the more number of mobile nodes that change their network attachment points in the interval t is. When r becomes small (i.e., t is enlarged), a message may visit a more number of mobile nodes whose states are stale in the network (see Figure 8(a)). In either case, the rudimentary implementation still introduces more unhelpful network traffic to the system (Figures 8(b) and (d)).



(a) $t_l = 5$ minutes (b) $t_l = 5$ minutes

(c) $t_l = 50$ minutes (d) $t_l = 50$ minutes

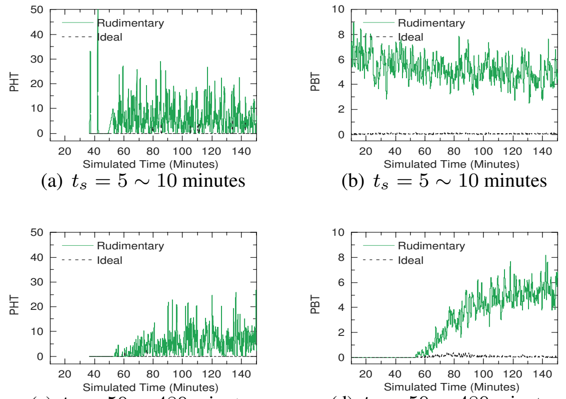
Figure 6. Effects of varying the peer lifetime



(a) $r = 0.03$ (b) $r = 0.03$

(c) $r = 3$ (d) $r = 3$

Figure 8. Effects of varying the request rate



(a) $t_s = 5 \sim 10$ minutes (b) $t_s = 5 \sim 10$ minutes

(c) $t_s = 50 \sim 480$ minutes (d) $t_s = 50 \sim 480$ minutes

Figure 7. Effects of varying the stationary period

5 Conclusions

We have presented the protocol design consideration for designing a mobile DHT and evaluated the performance for a rudimentary implementation. When compared with a likely ideal design, namely the IDEAL, which experiences near zero timeout without consuming additional network bandwidth in the presence of mobility churn, we conclude as follows.

- The rudimentary implementation has the fair performance results in terms of route delay if the system is operated under a high ordinary churn. That is, nodes in the systems have very short lifetime (e.g., five minutes).
- Also, the rudimentary implementation has the performance results similar to that of IDEAL when the route request rate is extremely high.
- The rudimentary implementation often generates unhelpful network traffics irrespective of different failure detection/recovery periods, peer lifetime, peer stationary periods and route request rates.

References

[1] H.-C. Hsiao and C.-T. King, "Tornado: A Capability-Aware Peer-to-Peer Storage Overlay," *Journal of Parallel and Distributed Computing*, vol. 64, no. 6, pp. 747–758, June 2004.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, August 2001, pp. 161–172.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 161–172, November 2001.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, August 2001, pp. 149–160.

[5] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz., "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[6] Y. C. Hu, S. M. Das, and H. Pucha, "Exploiting the Synergy between Peer-to-Peer and Mobile Ad Hoc Networks," in *Proceedings of the International Workshop on Hot Topics in Operating Systems*. USENIX, May 2003.

[7] H.-C. Hsiao and C.-T. King, *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*. CRC press, 2004 (to appear), ch. State Management in DHT with Last-Mile Wireless Extension, (a preliminary version appears in *International Parallel and Distributed Processing Symposium*, April 2003).

[8] A. Gupta, B. Liskov, and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays," in *Proceedings of the International Workshop on Hot Topics in Operating Systems*. USENIX, May 2003.

[9] H. Weatherspoon and J. Kubiatowicz, "Efficient Heartbeats and Repair of Softstate in Decentralized Object Location and Routing Systems," in *SIGOPS European Workshop*. ACM Press, September 2002.

[10] H.-C. Hsiao and C.-T. King, "Mobility Churn in DHTs," Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan, Tech. Rep., September 2004.