UnoHop: Efficient Distributed Hash Table with *O*(1) Lookup Performance

Herry Sitepu, Carmadi Machbub, Armein Z. R. Langi & Suhono H. Supangkat

School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Jl. Ganesha 10 Bandung, 40132 Indonesia

Abstract. Distributed Hash Tables (DHTs) with *O*(1) lookup performance strive to minimize the maintenance traffic required for disseminating membership changes information (events). These events dissemination allows each node in the peer-to-peer network maintains accurate routing tables with complete membership information. We present UnoHop, a novel DHT protocol with *O*(1) lookup performance. The protocol uses an efficient mechanism to distribute events through a dissemination tree that is constructed dynamically rooted at the node that detects the events. Our protocol produces symmetric bandwidth usage at all nodes while decreasing the events propagation delay.

Keywords: *Distributed Hash Table; peer-to-peer network; one-hop lookup protocol.*

1 Introduction

DHT designers employ two approaches when choosing the size of the routing table maintained by each node. In the first approach, each node maintains minimal information about its neighbors in the routing table while preserving the lookup correctness. Lookup for a key requires contacting some nodes along the path to the node responsible for the key. This multi-hop lookup increases the overall lookup latency.

The reason behind this approach is to minimize the communication cost required by each node to maintain accurate routing state entries in the rapid occurrences of membership changes. Some DHT protocols that use this approach are Chord [8], Tapestry [9], Pastry [13], and Kademlia [6]. All these DHTs maintain $O(\log N)$ routing states and are able to resolve a lookup via $O(\log N)$ messages to other nodes.

In the second approach, the designers argue that when the membership changes (*churn*) rate is not too high it is more efficient for each node to maintain complete membership information [14]. The benefit that comes with this design choice is that each lookup for a key can be resolved by contacting only one node. This one-hop lookup can be achieved by direct communication with the node that is responsible for the key. When compared with DHTs that use the

first approach, DHTs with *O*(1) lookup performance consume more communication bandwidth for maintaining the accurate routing entries [5].

When nodes join and leave the network rapidly, the routing entries in other nodes become inaccurate. To maintain high rate of successful one-hop lookup, each node requires a mechanism to detect events and propagate events to all other nodes in the network. The DHTs that use this approach are OneHop [4], Kelips [11], GRBM [15], and the protocol proposed by Leong et al. [12].

Application designers must consider these two approaches and the reasoning behind the design choice before choosing one DHT protocol as the overlay network layer for their particular application. The important considerations that one must take into account are how big the network will grow (total number of nodes) and the dynamic behavior of the nodes as defined by the churn rate in the system. When the churn rate is low then it is preferable to use DHT protocol with complete membership information [14].

The main problem that has to be addressed by DHTs with *O*(1) lookup performance is how to distribute membership events efficiently. When a node detects that an event has occurred, it must notify other segments of the network about the event, thus the routing table entries in all nodes can be updated and corrected according to the network latest condition.

In this paper, we propose one-hop DHT protocol that uses the dynamic dissemination tree that is constructed rooted at the node that detects the event. Each node at each level of the tree represents a segment of the network and has the responsibility to distribute the event to all nodes in its segment. When a node want to forward data to the next hop, the protocol employs Proximity Route Selection [3] to choose the node with lowest communication latency from the node.

We compare our protocol implementation with OneHop using p2psim simulator [5]. This is simply because OneHop is already integrated with p2psim. The simulation results show the improvement over OneHop's [4] lookup performance. Compared to OneHop, UnoHop has faster distribution delay especially for join events. OneHop bandwidth usage is asymmetric because slice leaders and unit leaders consume more bandwidth (5-10 times) than ordinary nodes. We show that by distributing the responsibility to propagate events to the nodes that detect the event will result in symmetric bandwidth consumption.

The rest of this paper is structured as follows. Section 2 describes some previous work related to one-hop Distributed Hash Table (DHT). Section 3

presents the UnoHop protocol. Section 4 describes the environmental setup for the simulation, and in Section 5 we show and discuss the simulation results. Finally, we conclude in Section 6.

2 Related Work

Gupta et al. [4] presented a novel algorithm that allows each node maintains accurate routing tables with complete membership information. They employ a three-level hierarchical dissemination system that quickly enough propagates membership changes information. The ring identifier is divided into k equal contiguous intervals called slices. Each slice has a fix slice leader, which is the successor of the center of the slice's interval. Each slice is divided into u equal contiguos intervals called units. Each unit has a fix unit leader, which is the successor of the center of the unit's interval. Each node runs stabilization procedure by sending ping messages prediodically to its predecessor and successor nodes at a predefined interval. Through this procedure each node can detects membership changes around its predecessor and successor nodes. When a node detects an event, it sends the event to its slice leader. The slice leader aggregates events from its slice for a time interval. Next, the slice leader sends the aggregated events to all other slice leaders. The received events are aggregated again for a time interval. Next, the slice leader dispatches the aggregated events to all its unit leaders. The unit leaders piggyback the events on the ping messages to their predecessor and successor nodes. The events are propagated to all nodes in a unit in one direction and stop when reaching the unit boundary.

Leong et al. [12] presented token-passing mechanism for distributing membership changes. The events detected by a node are distributed by constructing a token that contains the events. When a node receives a token, it can decide to send the token to its predecessor or generate q secondary tokens. The decision depends on the policy adopted and the available resources. If all nodes decide to generate q secondary tokens then each token will require $\log_q n$ hop to be succesfully propagated to all n nodes in the network. The tokens can be merged or destroyed under several conditions.

Wang et al. [15] presented Grouped Random Broadcast Messages protocol for maintaining global lookup table. Events are merged into notification message in the dispatcher node. Then notification messages are sent to all nodes using their event dissemination algorithm.

3 The UnoHop Protocol

UnoHop computes node's identifier by hashing its IP address using cryptographic hash function SHA-1 [1] into 128-bit length integer value. Data is represented with a key whose identifier is calculated using the same hash function. Both key identifiers and node identifiers are ordered on the same identifier space within a circle of modulo 2^{128} and creating ring-like overlay network topology. A key *k* is assigned to its successor (Figure 1), which is the first node with identifier value equals to or follows *k* in the identifier circle. Each node maintains information about its successor and predecessor.

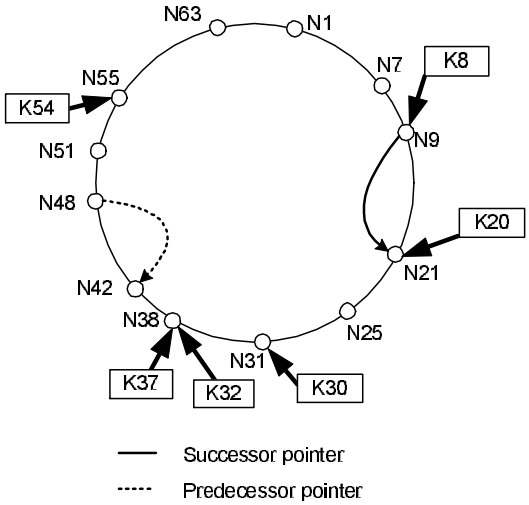


Figure 1 Identifier ring with m=6 and 12 nodes.

If node *n* wants to join the system, first it must contact the well-known node *w* that is already in the system and copies all the routing states from *w*. From the received membership information, node *n* knows its predecessor and successor and informing them about its presence. This procedure is similar to the Chord protocol [8].

Each node runs stabilization procedure by sending ping message periodically to its predecessor and successor at a predefined time interval (t_{stab}). Using this procedure, a node can know about the membership changes around its predecessor and successor nodes. When newly joined node *n* sends ping message to its successor *s*, node *s* checks its routing table whether *n* is its correct predecessor. If *s* finds that *n* is not its correct predecessor, than it adds *n* to its routing table. If a node does not receive any reply from its predecessor or

successor, then it knows that its predecessor or successor has leaved the network.

The membership changes must be propagated to all nodes in all segments of the ring to allow each node maintains correct and complete membership information in its routing table. UnoHop uses a dynamic dissemination hierarchy that is rooted at the node that detects the event. UnoHop divides the identifier ring into *k* equal and contiguous segments called slices. Each slice is further divided into *u* equal contiguous segments called units. Each slice is lead by a node that acts as the slice leader and each unit is led by a node that acts as the unit leader. UnoHop does not choose a fix slice leaders and unit leaders. Instead, it chooses them dynamically at the occurrences of events.

A node *n* that detects membership changes notifies all the slice leaders and sends the events notification to them. Node *n* selects the slice leader s_i from all nodes in slice *i* that is the *closest* from *n*. Next, slice leader s_i dispatches the events to all unit leaders in its slice. Slice leader s_i selects the unit leader u_j from all nodes in unit *j* that is the *closest* from s_i. Finally, when unit leader u_j runs its stabilization procedure, it piggybacks the events on the ping messages to the predecessor and successor. Each node that runs the stabilization procedure forwards the events to its predecessor or successor in one direction. If a node receives events from its successor, it piggybacks the events on the ping message to its predecessor. Otherwise, it piggybacks the events on the ping message to its successor.

The mechanism to distribute events is depicted in Figure 2. The network in Figure 2 contains 10 nodes. The identifier ring is divided into 3 slices. Each slice is further divided into 3 units. When a node *n* detects an event (node *f* has leaved the network) by running the stabilization procedure (step 1), it notifies all other segments about the event including its own segment. To distribute the event to its own slice, it selects a node that acts as the slice leader that will further distribute the event in its slice. The slice leader is selected as the closest node in a slice from node *n*. After *n* found a node that acts as its slice leader, it sends the event to that slice leader. To propagate the event to other slices, node *n* chooses the slice leader of a slice as the closest node in the slice from *n* and sends the event to all selected slice leaders (step 2).

Upon receiving an event notification, a slice leader dispatches the event to its entire unit leaders. To do this, a slice leader selects a unit leader as the closest node in a unit from it and sends the event to all selected unit leaders (step 3). A unit leader aggregates all events it receives. When a unit leader runs the stabilization procedure, it piggybacks the events on the ping message to its predecessor and successor (step 4). Finally, the events then piggyback one ping

message from one node to other nodes in one direction until it reaches the unit border. If a node receives the events from its predecessor then it will piggyback the events on the ping message to the successor. Otherwise, it piggybacks the events on the ping message to the predecessor. This mechanism avoids duplication in communication traffic.

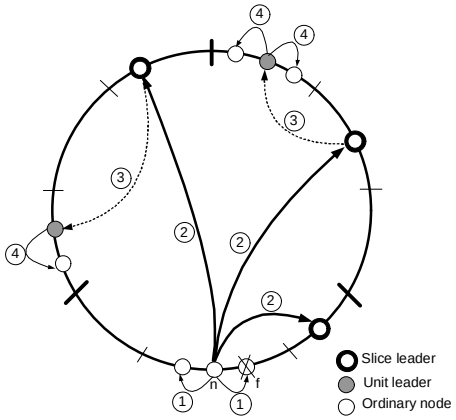


Figure 2 Hierarchical events distribution.

3.1 Characterizing Closeness

Our algorithm depends on the definition of closeness between nodes. The simple approach for characterizing closeness between nodes is the communication latency. A slice leader is selected as the node in a slice with the lowest latency from the node that detects an event. A unit leader is selected as the node in the unit with the lowest latency from its slice leader. We will describe several criteria that we used to characterize closeness between nodes.

Recent research uses the probability p of a neighbor being alive for characterizing the freshness of entries in the routing table [10]. In real systems, the distribution of node lifetimes follows a heavy-tailed Pareto distribution, where nodes that have been alive for a long time are more likely to stay alive for an even longer time. The probability of a node dying before time t is:

$$\Pr(lifetime < t) = 1 - \left(\frac{\beta}{t}\right)^{\alpha} \quad (1)$$

To simplify our analysis we set the scale parameter α with 1 (rounded from 0.83) and the shape parameter β with 1560 sec. Those values come from the results described by Saroiu et al. when studying the Gnutella network [7]. In

heavy-tailed Pareto distribution, a node lifetime distribution is estimated using both how long the node has been in the network Δt_{alive} and the time duration the node entry exists in the routing table Δt_{entry}. The conditional probability p_i of a neighbor i being alive as seen by node n [10]:

$$\begin{aligned} p_i &= \Pr\left(lifetime > (\Delta t_{alive} + \Delta t_{entry}) \middle| lifetime > \Delta t_{alive}\right) \\ &= \frac{\Pr(lifetime > (\Delta t_{alive} + \Delta t_{entry}))}{\Pr(lifetime > \Delta t_{alive})} \\ &= \frac{1 - \left(1 - \dfrac{\beta}{\Delta t_{alive} + \Delta t_{entry})}\right)}{1 - \left(1 - \dfrac{\beta}{\Delta t_{alive}}\right)} \\ &= \frac{\dfrac{\beta}{\Delta t_{alive} + \Delta t_{entry}}}{\dfrac{\beta}{\Delta t_{alive}}} \\ &= \frac{\Delta t_{entry}}{\Delta t_{alive} + \Delta t_{entry}} \end{aligned} \tag{2}$$

We use the probability of a node being alive as one of the criteria for characterizing closeness. This leads to the selection of nodes that have the large p_i, thus decreasing the probability of forwarding events to dead nodes.

When node n wants to forward the events to a slice leader s_y of slice y, node n selects a node i from all nodes in the slice y with the largest value of w_i:

$$w_i = \frac{p_i}{delay(n,i)} \tag{3}$$

where p_i is the probability of node i being alive and $delay(n, i)$ is the communication latency from node n to node i.

When slice leader s_y wants to dispatch the events to a unit leader l_z of unit z, node s_y selects a node j from all nodes in the unit z with the largest value of v_j:

$$v_j = \frac{p_j}{d(m_z, j) \cdot delay(s_y, j)} \tag{4}$$

where p_j is the probability of node j being alive, $delay(s_y,j)$ is the latency from slice leader s_y to node j, and $d(m_z,j)$ is the identifier distance from node j to the center of the unit z (m_z).

```
procedure find_slice_leader(y)
n_start = y * slice_size
n_end = (y + 1) * slice_size
n = successor(n_start)
w = 0
while n != n_end and n > n_start and n < n_end
   p_i = t_alive/(t_alive + t_entry)
   w_i = p_i /latency(me, n)
   if w_i >= w
      w = w_i
      slice_leader = n
  n = next(n)
return slice_leader
```

Figure 3 Psudo-code for finding a slice leader in a slice.

```
procedure find_unit_leader(y, z)
n_start = y * slice_size + z * unit_size
n_end = y * slice_size  + (z + 1) * unit_size
m_z = successor(y * slice_size + z * unit_size + unit_size/2)
n = successor(n_start)
w = 0
while n != n_end and n > n_start and n < n_end
   p_j = t_alive/(t_alive + t_since)
   d = |m_z - n|
   w_i = p_j / (d * latency(me, n))
   if w_i >= w
      w = w_i
      unit_leader = n
  n = next(n)
return unit_leader
```

Figure 4 Psudo-code for finding a unit leader in a unit.

Equation 3 shows that the algorithm prefers a node that is closest to the center of a unit as the unit leader. This is because the propagation in a unit starts from the unit leader and is directed to 2 directions, its predecessor and successor. If a unit leader is close to the center of a unit, then the communication cost required

for propagating the events can be decreased. Figure 3 and Figure 4 show the pseudo-code for finding the slice leader and the unit leader respectively.

4 Experimental Setup

We use p2psim [5] packet level simulator to simulate both UnoHop and OneHop algorithms. For fair comparison, we use 10 slices and 5 units for both UnoHop and OneHop. The stabilization interval for both algorithms are chosen as 1 second. We use some network topologies for the simulation. First, our simulation uses network that consists of 1024 nodes. The latency matrix of 1024 nodes are collected using King technique that uses recursive DNS queries to approximate the latency between nodes [3].

Second, our simulation uses network that consists of 400, 600, and 3000 nodes where each node has the Euclidean coordinate chosen randomly. The latency between nodes are calculated as the Euclidean distance between nodes. Both network topologies have round trip delay with mean 178 ms. We also generate lookup events for each node to lookup random key at a randomly chosen interval. The lookup interval is calculated by following exponential distribution with mean 1 lookup per second.

5 Simulation Results

To show how efficient the events distribution mechanism used by both algorithms when some nodes crash, we generate crash events at 200 seconds from the start of the simulation, triggering 40% of the nodes in the network that are chosen randomly to leave the network (crashed). One node crashed at a time every 1 ms. For the network topology consisting of 1024 nodes, all 40% randomly chosen nodes completely crashed after 410 ms. After the nodes crash, all lookup to those nodes will fail, increasing failure rate of the system and decreasing lookup performance.

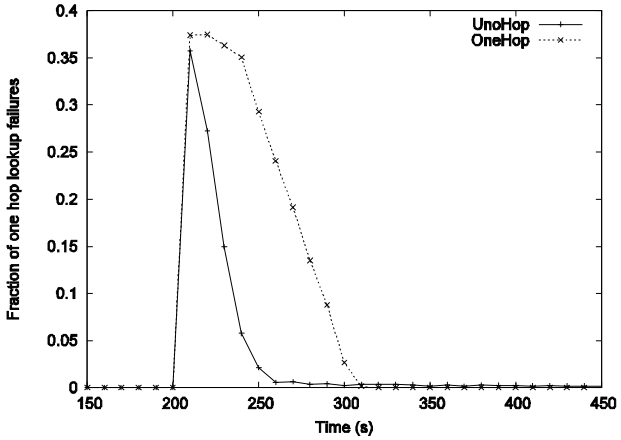


Figure 5 Lookup failure rate comparisons after crashing 40% nodes.

Figure 5 shows the failure rate of both OneHop and UnoHop at interval 180 second to 450 second. The graphic shows that our algorithm decreases the lookup failure rate faster then OneHop. For example at time 250 second (50 seconds after the crashes) the failure rate of OneHop still in 0.292 (about 30% of one-hop lookup have been failed) while our algorithm already reaches 0.038 (about 4% of one-hop lookup have been failed). OneHop reaches lookup failure rate back to around 0% after 100 seconds after the crashes while UnoHop reaches faster after 70 seconds.

Figure 6 shows the average of maintainance bandwidth usage per node for OneHop and UnoHop. The figure shows the bandwidth usage of leaders and ordinary nodes with OneHop algorithm compared with nodes with UnoHop algorithms. Because our algorithm does not choose the specific leaders then the bandwidth usage is symmetric between nodes.

Figure 6 shows that after 10 seconds UnoHop's nodes bandwidth usage increases rapidly. The reason is because our algorithm choose dynamic dissemination tree rooted at the node that detects the events and directly distributes the events to other nodes that represent slices, which further distribute events to all nodes in their slices. Nodes that act as leaders in OneHop algorithms consume 6-10 times bandwidth than ordinary nodes, this asymmetric behavior is problematic because special nodes with higher bandwidth capacity are required to become leaders.

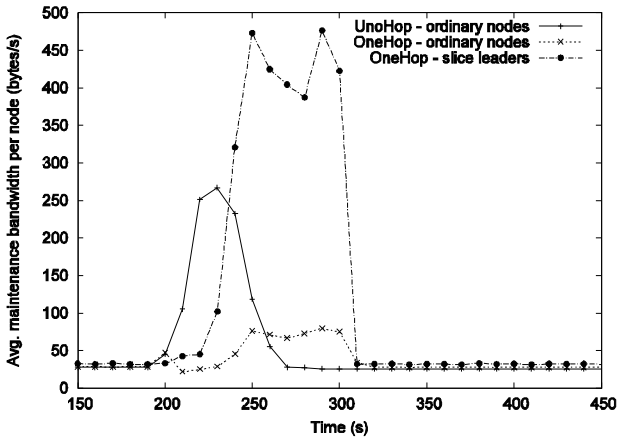


Figure 6 Average bandwidth usage of OneHop's slice leaders and ordinary nodes compared with UnoHop's ordinary nodes after crashing 40% nodes.

To show how efficient each algorithm when distributing join events, we generate rejoin events for previously crashed nodes. One node is rejoined at a time with interval 1 ms at time around 1000 second. After rejoin, there are also some movement of the keys. Some keys are reassigned to the newly joined nodes. Because all other nodes still have old entries in the routing table and do not know about newly joined nodes, then a fraction of one-hop lookup failure will increase again.

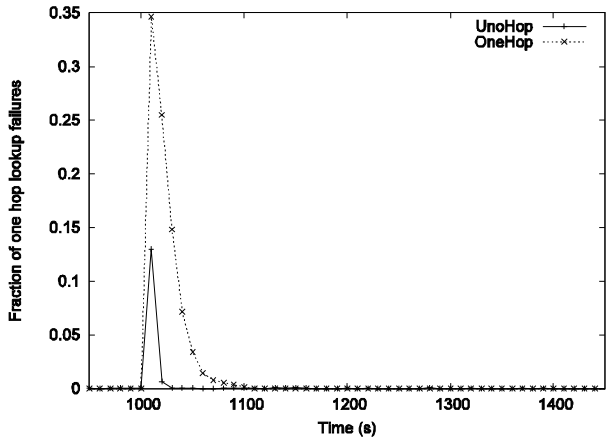


Figure 7 Lookup failure rate comparison after rejoining previously crashed 40% of nodes.

Figure 7 shows that UnoHop has significant performance improvement over OneHop when there are lots of join events. Each time a node joins the network, it also notifies all the slice leaders to distribute its join event. With this mechanism, the one-hop lookup failure rate decreases and returns faster back to almost 0%. Figure 8 shows the bandwidth usage of OneHop's slice leaders and ordinary nodes compared with UnoHop's ordinary nodes. It shows that the UnoHop's nodes consume much less bandwidth when distributing join events.

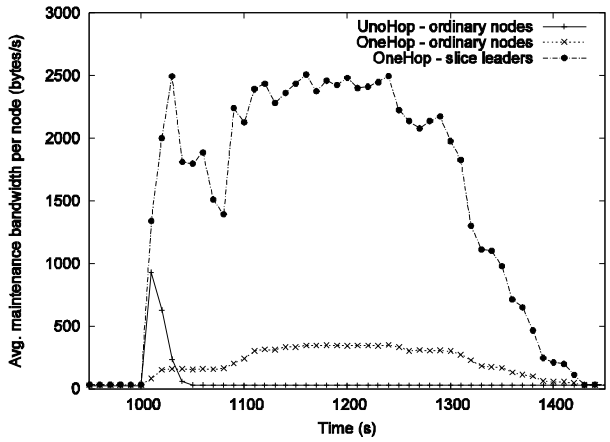
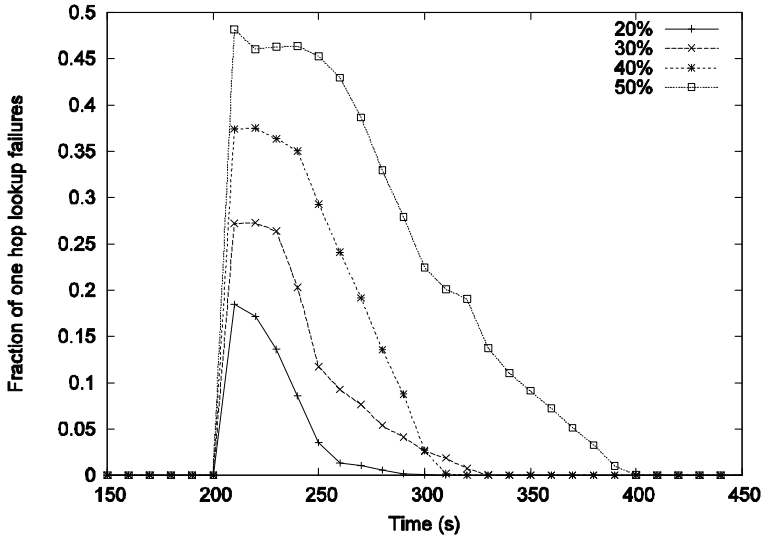


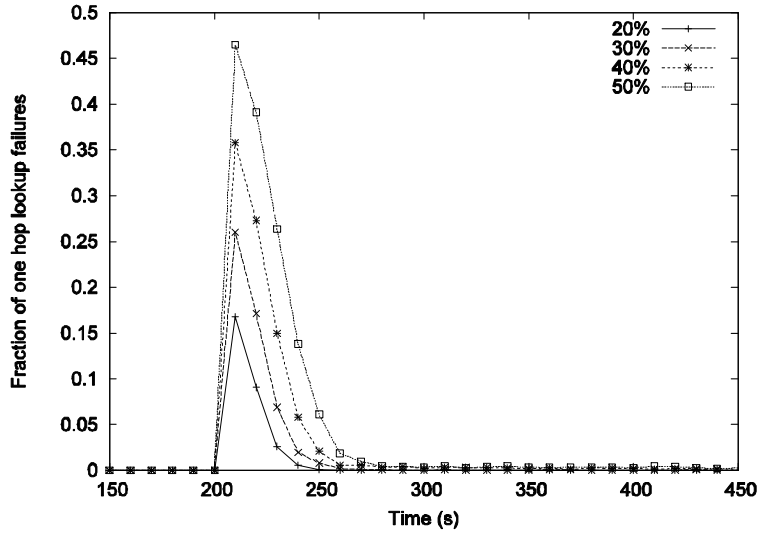
Figure 8 Average bandwidth usage of OneHop's slice leaders and ordinary nodes compared with UnoHop's ordinary nodes after rejoin of 40% nodes.

Figure 9 shows the one-hop lookup failure rates of OneHop and UnoHop from the simulation with different number of crashed nodes. In this simulation we generate different crash events for 20%, 30%, 40%, and 50% nodes. The increase of fraction of one-hop lookup failure rate is proportional with the number of crashed nodes. The lookup performance of UnoHop (Figure 9b) protocol shows slighter variation when compared with OneHop (Figure 9a) that has more distribution delay when the crashes are increased.

Figure 10 shows the one-hop lookup failure rates from the simulation with different number of nodes (network size). In this simulation we use networks with 400, 600, 1024, and 3000 nodes. The simulation results show that UnoHop (Figure 10b) has slighter variation when the network size increases. This is different from OneHop (Figure 10a) where the performance degrades significantly as the network size is increased.

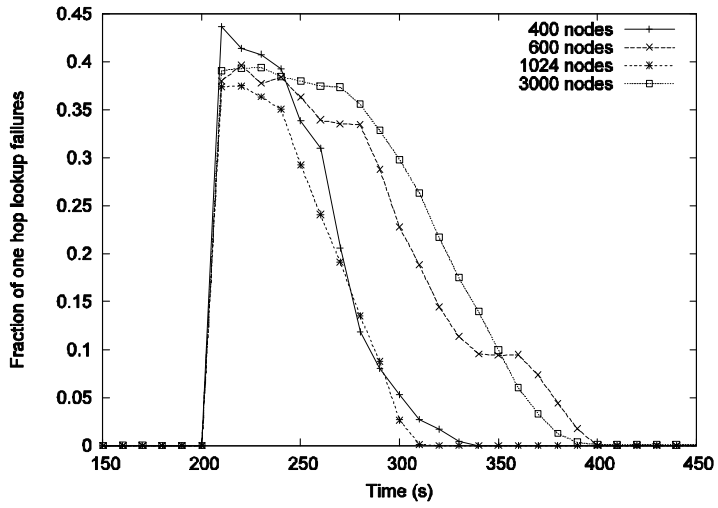


(a) OneHop

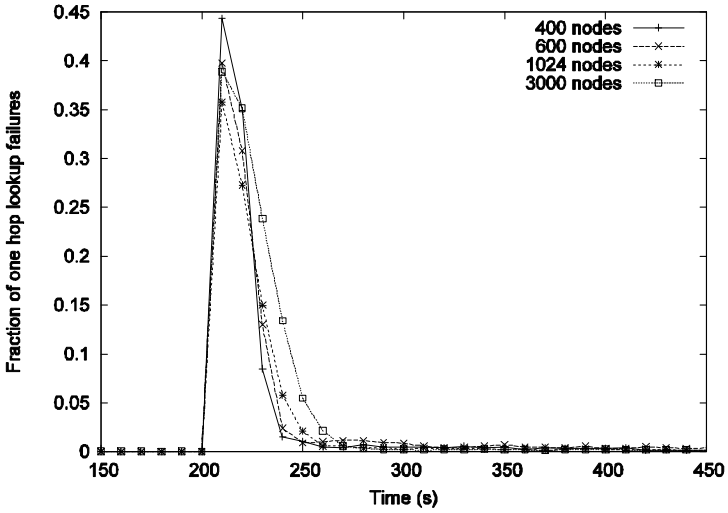


(b) UnoHop

Figure 9 Comparison of different crashes.



(a) OneHop



(b) UnoHop

Figure 10 Comparison of different network sizes.

6 Conclusion

DHT algorithms that use routing table with complete membership information are suitable for applications that have low membership changes rate and with moderate network size up to thousands nodes. Fixed dissemination tree used by OneHop will produce asymmetric bandwidth usage among slice and unit leaders when compared with ordinary nodes. The randomly chosen leaders must have the required bandwidth to distribute events. The communication bandwidth consumed by leaders will grow proportionally with the size of the network.

Our algorithm employs proximity route selection to construct the dissemination tree dynamically rooted at the node that detects the events. This technique avoids the asymmetric bandwidth usage at slice leaders as the result of OneHop's design for using fixed dissemination tree. The simulation results show that as the networks grow bigger, UnoHop outperforms OneHop significantly. The improvement is shown by faster events distribution delay of our algorithm.

References

[1] FIPS180-1. *Secure hash standard, U.S. Departement of Commerce*, April 1995.

[2] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S. & Stoica, I., *The impact of DHT routing geometry on resilience and proximity*, In SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 381–394, New York, NY, USA, ACM Press, 2003.

[3] Gummadi, K.P., Saroiu, S. & Gribble, S.D. *King: Estimating latency between arbitrary internet end hosts*, In Proceedings of the SIGCOMM Internet Measurement Workshop (IMW 2002), Marseille, France, November 2002.

[4] Gupta, A., Liskov, B. & Rodrigues, R., *Efficient routing for peer-to-peer overlays*, In Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04), March 2004.

[5] Li, J., Stribling, J., Morris, R., Kaashoek, M.F. & Gil, T.M., *A performance vs. cost framework for evaluating dht design tradeoffs under churn*, In Proceedings of the 24th INFOCOM, Miami, FL, March 2005.

[6] Maymounkov, P. & Maziã¨res, D., *Kademlia: A peer-to-peer information system based on the XOR metric*, In Proceedings of the First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002. Revised Papers, pp. 53–65, 2002.

[7] Saroiu, S., Goummadi, P. & Gribble, S., *A measurement study of peer-to-peer file sharing systems*. In Proceedings of Multimedia Computing and Networking (MMCN), San Jose, CA, USA, January 2002.

[8] Stoica, I., Morris, R., Karger, D., Kaashoek, F.M. & Balakrishnan, H., *Chord: A scalable peer-to-peer lookup service for internet applications*, In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, **31**, pp. 149–160, New York, NY, USA, ACM Press, October 2001.

[9] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. & Kubiatowicz, J.D., *Tapestry: A resilient global-scale overlay for service deployment*, IEEE Journal on Selected Areas in Communications, **22**(1):41–53, 2004.

[10] Li, J., Stribling, J., Morris, R. & Kaashoek, M.F., *Bandwidth-efficient management of DHT routing tables*, In Proceedings of the 2nd Symposium on Networked System Design and Implementation (NSDI'05). May 2005.

[11] Gupta, I., Birman, K., Linga, P., Demers, A. & van Renesse, R., *Kelips: Building an efficient and stable p2p DHT through increased memory and background overhead*. In IPTPS '03: 2nd International Workshop on Peer-to-Peer Systems, 2003

[12] Leong, B. & Li, J., *Achieving one-hop DHT lookup and strong stabilization by passing tokens*. In Proceedings of the 12th International Conference on Networks 2004 (ICON 2004), Singapore, November 2004.

[13] Rowstron, A. & Druschel, P., *Pastry: Scalable, Decentralized Object Location and Routing for Large Scale Peer-to-Peer Systems*, In International Conference on Distributed Systems Platforms (Middleware), pp. 329-350, Heidelberg, Germany, November 2001.

[14] Rodrigues, R. & Blake, C., *When Multi-Hop Peer-to-Peer Lookup Matters*, In Proceedings of IPTS, San Diego, CA, USA, February, 2004

[15] Wang, W., Wang, G., Liu, X., Liu, J., *One-Hop DHT Lookup based on Grouped Random Broadcast Messages*, In Proceedings of the 2007 IEEE International Conference on Integration Technology, Shenzen, China, March 2007.