International Journal of Grid and Distributed Computing
Vol. 9, No. 7 (2016), pp.1-12
http://dx.doi.org/10.14257/ijgdc.2016.9.7.01

An Efficient DHT Routing Protocol with Small-world Features for Structured P2P Network

Bin Zeng[1] and Rui Wang[2]

[1]*Department of Management Engineering, University of naval Engineering, China*
[2]*University Library, Department of Training,*
University of naval Engineering, China

Abstract

Distributed Hash Tables (DHT) provide a fault-tolerant and scalable means to store data blocks in P2P systems. In this thesis, we have proposed an improved version of CAN called "small-world DHT". The key idea is to link each CAN node to a constant number of long-distance contacts, which are chosen according to the probability density function. This enables our scheme to achieve O(log^{2}n) routing path length with O(1) routing table size per node. We have shown that the hybrid infrastructure of a structured overlay network (CAN) and a random graph (small-world model) not only preserve CAN's simplicity, but also achieves a resilient and efficient DHT routing algorithm. We have also exploited the LookAhead-GREEDY routing algorithm, whereby each node obtains information about its neighbors' neighbors by periodically exchanging the routing table entries with its neighbors. The routing algorithm improves the routing efficiency and enhances the network's resilience to failure. Our scheme requires knowledge of the current network size in order to construct the long-distance links. To estimate the network size, the latter maintains a distributed binary partition tree and measures the size of the sample area by counting the number of leaves in the corresponding branch.

Keywords*: Distributed Hash Tables*; *Structured Overlay Network*; *Routing Algorithm*

1. Introduction

In recent years, peer-to-peer overlay networks have gained popularity due to the success of file sharing applications [1, 2]. Most popular file-sharing systems store the mapping table at a central server, and each node consults the central server to obtain the location of an object. However, the server could become a single point of failure or a performance bottleneck due to the short lifetime of hosts. For this reason, Distributed Hash Tables (DHTs) that do not need central servers have emerged, where users are allowed to insert, remove, and lookup some objects stored in another place through a distributed hashing function. The object identifier is mapped to a partition managed by some node in the overlay network. Hash lookup for locating certain objects is relayed appropriately via some routing algorithm to the manager that possesses the corresponding table entry. If the network size is of the order of hundreds or thousands of hosts, and if hosts arrive and leave frequently, maintaining the mapping table in a distributed manner might be challenging. DHT routing topologies have two conflicting goals: fast lookup and small state. Some schemes proposed recently have achieved an optimal trade-off.

On the other hand, the small-world phenomenon[3] found in many large interconnected systems possesses a paradoxical ability in that any individual is only a short sequence of acquaintances away from any other peer in the system[4]. Kleinberg proposed an algorithmic construction comprising a family of random graphs that allow an average $O(\log^2 n)$ path length with only $O(1)$ links per node using greedy routing.

Symphony[5], a DHT for full lookups, was inspired by Kleinberg's construction. The key idea is to arrange all participants along a ring and equip them with long-distance

ISSN: 2005-4262 IJGDC

contacts drawn from a family of harmonic distributions. Symphony can be regarded as emulating a small-world model on Chord by modifying the manner of long-distance link construction. A similar idea is applied by [6]. To improve Freenet's performance, [7] proposed an enhanced-clustering cache replacement scheme based on a small-world model in place of LRU cache replacement, which has a poor hit ratio within creasing load. The new hybrid construction possesses both the anonymous functionality of Freenet and the superior cache replacement scheme from the small-world model. This prompts us to suggest that a relatively simple, yet efficient, randomized construction can be obtained by emulating Kleinberg's small-world networks on an existing DHT. Thus, we can concentrate on the design of long-distance links construction and routing topologies and leave the other details to the original protocol.

Because of the dynamic nature of peer-to-peer networks, a major challenge is how to maintain desirable small-world characteristics without losing the advantages of the original DHT. In short, it is important to ensure that the combined scheme achieves synergy. We prefer to choose a DHT which has advantages such as simplicity, fault-tolerance, and a small state but doesn't perform well in routing efficiency so that we can enhance it by emulating the small-world model on the DHT. A suitable choice is a 2-dimensional CAN, a scheme not only similar to the two-dimensional mesh used in Kleinberg's constructions, but also simple enough and good at dealing with the arrival and departure of nodes. In addition to the routing issue, unknown network size is another source of uncertainty that makes emulation of the small-world network challenging. We address this issue by developing an estimation protocol to estimate the current size of the network.

2. Protocol Overview

Let I denote a 2-dimensional [0, 1]×[0, 1] Cartesian coordinate space that wraps around on a 2-torus. The virtual coordinate space is used to store (key, value) pairs as follows: to store a pair (K_1 , V_1), key K_1 is mapped to a point, W , in I using a uniform hash function. The key-value pair is then stored at the node that manages the zone in which point W is located. When a new node P arrives, to join the overlay network, it should know at least one existing node as the entry point. It then chooses a point (x, y) in the virtual space uniformly at random. After that, it locates node M, the current manager of (x, y), using the lookup function provided by the entry point. Subsequently, M would splits its allocated zone in half, retaining half and handing the other half to P. The neighbors of the split zone are notified of the update so that routing can include the new node. In the next step, to obtain the network size n, P initiates an estimation protocol. It then constructs the long-distance link with a node, which is chosen using the probability density function p_n. Each node maintains $O(1)$ short-distance links with its immediate neighbors. Since all nodes choose their IDs uniformly from I, they are expected to have roughly equal-sized zones.

The underlying structure is almost the same as CAN. The major difference is that our new scheme equips each node with $O(1)$ long distance links, which are chosen by a probability distribution function we will define shortly. To give readers a general picture, we first demonstrate the most basic components of our design: the methods for constructing long distance links and the routing algorithms. We then discuss how to estimate network size and maintain the small-world property in a dynamic environment.

Suppose node P arrives and we want to construct a long distance link for it. Let (x_P, y_P) be the center point of the zone. Our design aims to ensure that any node Q in I has the probability of at least of being chosen by P which is denoted by Eq. (1), where n denotes the number of nodes in the overlay network.

$$P_n(x_Q, y_Q) = \begin{cases} \dfrac{1}{((x_P - x_Q)^2 + (y_p - y_Q)^2)\pi \ln \dfrac{n}{2}} ; \dfrac{\sqrt{2}}{2} \geq \sqrt{(x_P - x_Q)^2 + (y_p - y_Q)^2} \geq \dfrac{1}{\sqrt{n}} \\ 0 \qquad\qquad ; \text{otherwise} \end{cases} \quad (1)$$

Note that the x-axis represents the difference of x_Q and x_P, and the y-axis represents the difference of y_Q and y_P. In Equation (1), $(x_P - x_Q)^2 + (y_P - y_Q)^2$ equals the square of the distance between P and Q. (x_Q, y_Q) could be converted to the expression $(x_P + r\cos\theta, y_Q + r\sin\theta)$, where r is the radius and θ is the angle. According to this, we modify Eq. (1) slightly and obtain

$$P_n(r, \theta) = \frac{1}{(((x_P + r\cos\theta) - x_P)^2 + ((y_P + r\sin\theta) - y_P)^2 \pi \ln \dfrac{n}{2}} = \frac{1}{r^2 \pi \ln \dfrac{n}{2}}$$

In a similar way to Symphony, we use the inverse transformation method to facilitate the generation of the random bivariate (x, y), which conforms to the above probability distribution function. At first, we find the cumulative distribution function of r:

$$t = F(r) = \frac{\ln(r^2 n)}{\ln \dfrac{n}{2}} ; \frac{\sqrt{2}}{2} \geq r \geq \frac{1}{\sqrt{n}} \quad (2)$$

The function is continuous and strictly increasing when $0 < F(r) < 1$, so that we find an inverse function F^{-1}

$$r = F^{-1}(t) = \sqrt{\frac{e^{t \ln \frac{n}{2}}}{n}} ; 0 \leq t \leq 1 \quad (3)$$

Then the algorithm for generating a random bivariate (x, y) having the probability distribution function p_n is as follows:

1. Generate $t \sim U(0, 1)$;
2. Return $r = F^{-1}(t)$.
3. Produce a random number θ distributed as uniform [0, 1].
4. Derive $x = (x_P + r\cos(360^*\theta) + 1) \bmod 1$, $y = (y_P + r \sin(360^*\theta) + 1) \bmod 1$.

Here we use modulus 1 since the overlay network is based on a 2-torus with side length 1. Afterward, node P contacts the manager of the point (x, y) by following a routing protocol, which we describe as follows. Finally, P attempts to establish a link with the manager of (x, y).

3. Routing Algorithm

In a trivial greedy algorithm when a node wants to look up a resource, it has to contact the manager of the zone that contains the coordinate of the resource. Each node forwards the lookup request to one of its contacts, including short-distance and long-distance, which minimizes the distance to the target coordinate.

We introduce the LookAhead-GREEDY algorithm to improve the routing efficiency. The key idea is to let each pair of nodes connected by a link exchange their routing tables. Instead of forwarding messages greedily, the current message holder could "look ahead", choosing a neighbor's neighbor that is closest to the target. Since the virtual space in our scheme, 2-dimensional torus, is a special case of small-world percolation according to the definition given in [8], it is reasonable to show that the mean routing paths of our scheme are $O(\log n / \log\log n)$. The cost contains not only $O(d^2)$ memory space, where d represents the mean degree of each node and is asymptotically constant, but also maintenance. Due to the prevalence of inexpensive hardware and transient user populations in peer-to- peer networks, the second issue is undoubtedly much more important than the first. In *i*-look

ahead, which means the node can obtain knowledge of the node $i + 1$ hops away, the arrival and departure of any node needs $O(d^{(i+1)})$ messages to update other nodes' routing tables. Fortunately, in the original CAN, each node sends "keep-alive" messages periodically to each of its neighbors. Thus, we can exploit the "keep-alive" messages to send the update piggy-backed.

4. Estimation Method

The construction of long-distance links relies on knowledge of network size. However, it is difficult for each node to obtain the current network size, especially in a network with frequent arrivals and departures. A relatively trivial approach, proposed and adopted by Viceroy [9] and Symphony, is based on the following insight. By randomly choosing a set of s distinct nodes, we compute the sum of the zone areas managed by these nodes. The trivial approach does not guarantee how close the estimate will be to the network size.

We propose better approaches which could achieve the above requirements. By two different approaches with incremental considerations, we demonstrate how to estimate network size using some sampling techniques. The former employs the concept of "binary partition tree" of CAN. This approach provides an unbiased estimate by sacrificing a certain degree of load-balancing. Even so, the smooth tradeoff between the deviation of the estimate and the level of load balancing could be obtained, and participants could tune the configuration even at run-time. While the first approach may cause hot spots, the flooding approach, as implied by the name, aims to achieve a estimation scheme based on flooding. It also guarantees an upper bound on the estimation error and provides an even smaller error than the first approach. The drawback is that the maximum network size is limited to 2^{62}, however, this is large enough in practice.

4.1. Partition Tree Method

The key idea is from the "binary partition tree" of CAN, in which each current zone could be treated as a leaf. The internal vertices in the tree represent zones that no longer exist (*i.e.*, they have split into several smaller zones at some previous time). The children of a tree vertex are the two zones split from it. CAN doesn't maintain such a partition tree in practice, but only uses the concept to design its takeover algorithm. In our work, we develop the estimation scheme by emulating a binary partition tree based on CAN. Each node G simulates an internal vertex[1] and a leaf, which represents the zone managed by G, and each tree vertex is assigned a unique identifier. Here the node G maintains the following information of a zone:

1. The zone identifier, *e.g.*, "1000" for one of the zones managed by node 2,
2. A link to its parent, and two links to its children.

Consider a specific node with id x, which manages a leaf vertex v. Let n_i denote the number of vertices that share the top i bits with v. For example, if $x = 1010$, and the other nodes are 1011, 1100, and 1001, then $n_2 = 2$, and $n_3 = 1$.

It is suggested that node x could detect all the nodes whose top l bits equal x's. Because the sample size is $1/2^l$ of the population, we can derive the network size by scaling the observed density. To find the largest value of l, x begins at its allocated leaf vertex. l is set as the bit length of the leaf id. In Fig. 1, *e.g.*, node 9 runs the estimation protocol. It first checks how many leaf vertices share the top 4 bits with the vertex 0100, surely one in this case since the vertex is a leaf. If $1 \geqslant 16(1+\delta)\delta^{-2}\ln(2^4 \times 1)$, we obtain an unbiased estimate $2^l n_l$ for the network size.

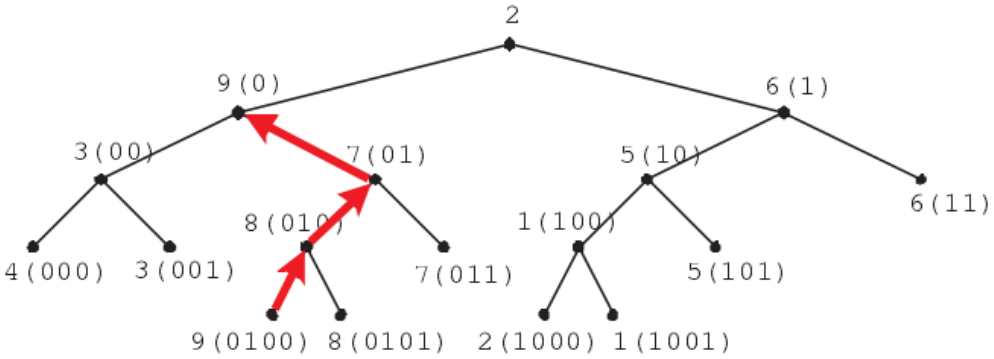

Figure 1. Routing of Finding the Largest Value of *l*.

If the condition isn't satisfied, *x* should try a smaller *l*. It would forward an estimation request to the node managing its parent vertex. Following the preceding example, node 9 will forward the request to node 8, because node 8 manages the vertex 010, which is the parent of 0100. The receiver of the estimation request would measure the n_l using a recursive detection algorithm.

The larger δ is, the larger the largest l which satisfies this condition is. In other words, the larger deviation the participant could tolerate, the more efficient the sampling procedure is. The design is not fully load-balancing since the node managing higher level vertex would be queried more frequently. Nevertheless, we can lower the degree of unbalance by loosing the deviation bound δ. This smooth trade-off makes it easy to use the mechanism as a module for other peer-to-peer applications. Users can tune the parameter δ depending on their requirement.

4.2. Flooding Method

Compared with the first approach, this approach is fairly simple and directly extended from the network size estimation scheme with minor adaptation to run on a 2-torus, the virtual space used by CAN. We describe the compute, estimate, and other subroutine called by these operations.

Algorithm 1: compute *estiID*
upon receiving <*computeEstiIDOp*, x_{left}, x_{right}, y_{up}, y_{down}>
$X_{10}= (x_{left} + x_{right})/2 *2^{31}$
$Y_{10:}= (y_{down} + y_{up})/ 2 *2^{31}$
convert X_{10} to a 31-digit binary X_2
convert Y_{10} to a 31-digit binary Y_2
append Y_2 to the least significant end of X_2
send <*computeEstiIDOp*, X_2> to startnode

The compute operation is used to compute the *estiID* for a node. As illustrated in Algorithm 1, *estiID* is determined by the coordinate of the zone center. The x-coordinate and y-coordinate of the zone center are converted separately to a 31-digits binary arrays, and then we concatenate the two binary arrays and obtain a 62-digits *estiID*. Since each node manages a unique zone, each node would obtain a unique *estiID* from this operation.

Algorithm 2 : estimate network size
upon receiving <*estimateOp*, x_{left}, x_{right}, δ, *estiID*> :
for l= 62 to 31 do
Y_2^{up}:= the smallest 31-digit binary number that shares the top l -31 bits with *estiID*[31,62]

Y_2^{down}:= the largest 31-digit binary number that shares the top l -31 bits with *estiID*[31,62]
convert Y_2^{up} to decimal Y_{10}^{up}
convert Y_2^{down} to decimal Y_{10}^{down}
Y_{up}:= Y_{10}^{up} /2^{31}
Y_{down}:= Y_{10}^{down} /2^{31}
X_{center}:= (x_{left}+ x_{right})/2
send <*measureOp*, X_{center}, Y_{up}, Y_{down}, *estiID*[0, l]> to local host
upon receiving <*measureOp*, n_l>
if $n_l \geqslant 16(1+\delta)\delta^{-2}\ln 2^l n_l$ then
send <*estimateOp*, $2^l n_l$ > to startnode
end if
end for
for l= 30 to 0 do
X_2^{left}:= the smallest 31-digit binary number that shares the top l bits with *estiID*
X_2^{right}:= the largest 31-digit binary number that shares the top l bits with *estiID*
convert X_2^{left} to decimal X_{10}^{left}
convert X_2^{right} to decimal X_{10}^{right}
X_{left}:= X_{10}^{left} /2^{31}
X_{right}:= X_{10}^{right} /2^{31}
send <*informAgentOp*, X_{left}, X_{right}, *estiID*[0, l]> to local host
upon receiving < *informAgentOp*, n_l>
if $n_l \geqslant 16(1+\delta)\delta^{-2}\ln 2^l n_l$ then
send <*estimateOp*, $2^l n_l$ > to startnode
end if
end for

As mentioned earlier, when a node *A* initiates the estimate operation (Algorithm 2), it would attempt to identify the largest such that $n_l \geqslant 16(1+\delta)\delta^{-2}\ln 2^l n_l$. It begins from l = 62 to 0, finding the first l which satisfies the condition. The procedure of measuring n_l could be divided into two phases: The first phase ranges from l = 62 to 31, and the second phase ranges from l = 30 to 0. In the first phase, the procedure measures the number of nodes which share the most significant i bits with *A*, $i \geqslant 31$. Since the top 31 bits of the *estiID* represents the *x*-coordinate of the allocated zone center, the search region is limited to the node whose allocated zone crosses the line $\{x = X_A\}$, where X_A denotes the x-coordinate of *A*'s allocated zone center. If the condition is not satisfied in the first phase, the operation will step into the next phase.

The second phase involves with larger searching space. Each iteration begins at calculating the left (X_{left}) and right (X_{right}) boundaries of the search region. Then the original node sends the *informleftOp* and *informrightOp* messages to its left and right neighbors. Subsequently, the recipients would forward the message to its neighbors in the same direction. This procedure would be repeated until reaching the boundary. All recipients, we call "agents", would count the number of matching nodes.

5. Dynamic Operations, Maintenance and Recovery

In a dynamic environment, the relatively frequent arrival and departure of nodes make the system behavior more unpredictable. We should ensure that the system works normally in the presence of a variety of failure. We will describe how we handle the following situations.

5.1. Node Joins

When a new node arrives, it changes certain nodes' state by some operations. If a lookup occurs before the whole operations has finished, it would encounter one of three scenarios. The common case is that the routing table entries involved in the lookup are still up-to-date, and the lookup reaches the target in $O(\log^2 n)$ hops. The second case is where short-distance links are correct, but long-distance links hasn't been constructed yet. This case doesn't affect the correctness but reduces the performance. In the final case, also the worse case, the lookup request is delivered to the node x while x is constructing its short-distance links, as an arrival, or assigning the short-distance links to its new neighbor, as an existing node. The incorrect short-distance links may lead to temporal disappearance of some keys since the node holding these keys cannot be reached. Once finding the desired resource was not found, the higher level application which uses small- world CAN as a module could send the lookup message again after a certain time period.

5.2. Node Failure and Leave

When a node *x* leaves the small-world DHT, we should ensure that its allocated zone is taken over by the remaining nodes, and the links to x are redirected to other effective nodes. In practice, our small-world DHT need to deal with nodes that fail or leave voluntarily. CAN's takeover algorithm has done lots of work for us in addition to the part relating to long-distance links and estimation mechanism, which make small-world DHT different from the original CAN. When a node x leaves the small-world DHT, we should ensure that its allocated zone is taken over by the remaining nodes, and the links to x are redirected to other effective nodes. In practice, our small-world DHT needs to deal with nodes that fail or leave voluntarily. CAN's takeover algorithm has done lots of work for us in addition to the part relating to long-distance links and estimation mechanism, which make small-world DHT different from the original CAN.

In order to deal with the impact of node failure on lookup, we employ the recovery method of CAN to deal with the adjustment of short-distance links and omit the details. The remaining problem is the reassignment of long-distance links. Compared with deterministic routing algorithm, our randomized scheme is relatively easy to maintain due to the loose-coupled structure of small-world DHT. While a node *x* fails, other nodes with long-distance link to *x* are not aware of the failure until attempting to forward some message through this link. Then they will rebuild these links using the above-mentioned construction algorithm, redirecting the long-distance link to another node which is chosen using the probability density function p_n. Because the small-world property merely relies on a random graph in which the probability of a long-distance link exists between any pair of nodes conforms to p_n, it's not as vulnerable as deterministic algorithm and more flexible and robust.

In order to deal with the impact of node failure on estimate mechanism, we proposed two versions of estimation mechanism. The partition tree approach is based on a binary partition tree, which obviously makes it more vulnerable to the failure. A similar distributed data structure called "binary search tree-like structure" [10] can emulate a binary tree on CAN. Several techniques to ensure the correctness of the BST-like structure were presented. These techniques could be applied to our partition tree approach.

In the flooding approach, we don't need to maintain an additional data structure but only build on the original overlay network. The advantage is that we needn't bother with extra maintenance. Since the overall messages produced by this approach are delivered upon short-distance links, the correctness would be ensured as long as the recovery mechanism of the original CAN works consistently.

For voluntary node leaves, since small-world DHT is robust in the presence of failure, a node voluntarily leaving the system could be regarded as a node failure. Nevertheless, we can improve the performance by using the following enhancement: when a node *x* leaves, it would notify the nodes with long-distance links to *x* before departure. Consequently, these nodes which are informed would rebuild their long-distance link with another remaining node.

5.3. Maintenance of Long-Distance Links

As mentioned earlier, the construction of long-distance links requires the knowledge of network size. With the frequent arrival and departure of nodes, the network size changes dramatically [11, 12]. The long-distance links constructed earlier gradually become stale. There- fore, even if there is no notification from leaving long-distance neighbors or failed connections detected, each node still needs to rebuild their long-distance links periodically in order to catch the current state of network. To save excessive traffic for constructing the long-distance link afresh, a compromised re-linking criterion could be used. We maintain a value $\check{n}_x$ as the estimate by node *x* at which its long-distance link was last constructed. Let $\check{n}'_x$ denote the current estimate obtained from the estimation mechanism running periodically. *x* only reconstructs its long-distance link if $\check{n}'_x$ deviates from $\check{n}$ too much, *e.g.*, $\check{n}'_x/\check{n} \notin [1/\delta, \delta]$, where δ is a user parameter.

6. Experimental Results

To study our small-world DHT's performance in the virtual space and the real topology, we designed a discrete event simulator that uses a centralized event queue to emulate the behavior of peer-to-peer systems on a single computer. The simulator consists of four major components: *entity*, *protocol controller*, *event scheduler*, and *statistics collector*. *Entity* maintains the basic information of a node, *e.g.*, node id, the range of an allocated zone, immediate neighbors and the long-distance contact. The major part of the routing algorithm is implemented in the protocol controller. Each *protocol controller* class, matching a unique protocol, shares a common interface so that we can change the underlying protocol easily with the same configuration while measuring the performance of different protocols. *Event scheduler* is responsible for creating and managing events. It is assumed that each event occurs in pseudo-parallel, *i.e.*, only one event occurs at any instance of real time, but many events may occur concurrently at any instance of simulation time. In the whole process, *statistic collector* is used to collect the statistical information. After the simulation is completed, the data will be processed and analyzed. The simulator allows users interacting with the system, *e.g.*, insert nodes, get mean routing hops, or get each node's estimate of network size. The metrics we use to evaluate the routing performance contains routing efficiency, lookup stretch, degree of load balancing, and resilience to failure. For simplicity, it is assumed that all nodes in the overlay network know the accurate network size.

Symphony was the first work that tried to implement a small-world model on a ring with unit perimeter. Here we examine the two models' performances by comparing their mean routing hops per lookup with their *k* long-distance links. Fig.2 clearly shows that: (1) increasing the number of long-distance links from 1 to 2 reduces the mean routing path length significantly. However, the marginal utility of successive additions diminishes. (2) The network with *k* random long-distance links does not expand as rapidly as the one demonstrated in Symphony, in which the path length grows by $O\sqrt{n/k}$; nevertheless, it still does not scale well compared to the small-world network.

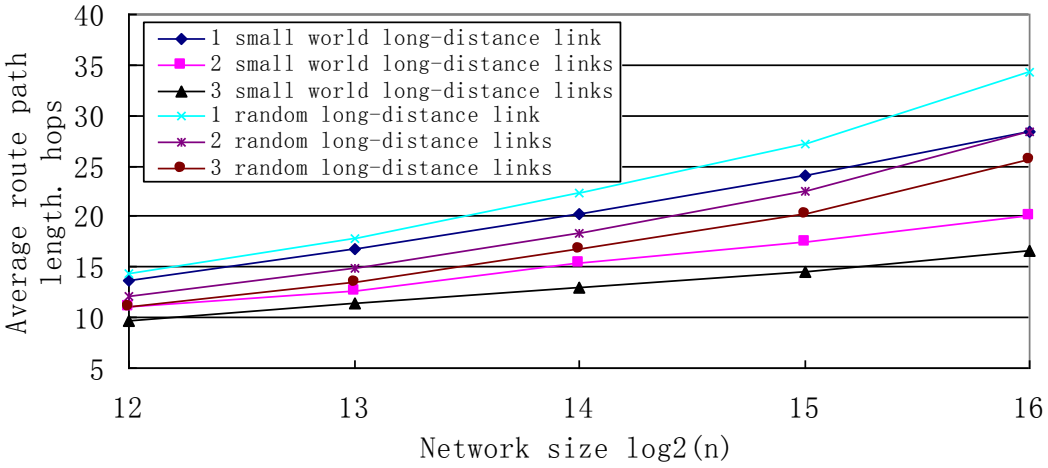

Figure 2. Comparison with a Network where each Node Links to *k* other Nodes Chosen Uniformly at Random

Fig.3 shows that the unidirectional Symphony has a larger mean routing path length in the respective network scale than the small-world DHT; however, it is still bounded by $O(\log^2 n)$. Besides, the bidirectional Symphony improves the routing efficiency significantly and makes the mean routing path length tight with that of small-world DHT.

Although the two networks have similar routing path length on the respective virtual space, they may achieve different mean lookup latency since the messages may travel arbitrarily long distances in the Internet in each virtual routing hop. To compare the small-world network with bidirectional Symphony, both equipped with 2 long-distance links, we measure the routing latency on physical networks by calculating the sum of the physical link weight along the physical routing path. It is assumed that the physical routing is based on the shortest path algorithm. Fig.4 clearly shows that, in each network scale, our scheme achieves lower lookup latency on the underlying topology.

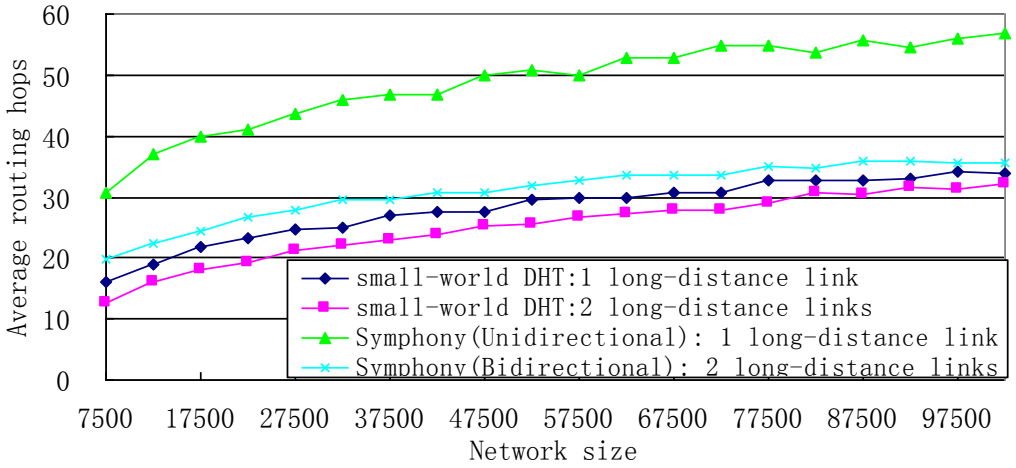

Figure 3. Mean Routing Hops for the Small-World DHT and Symphony

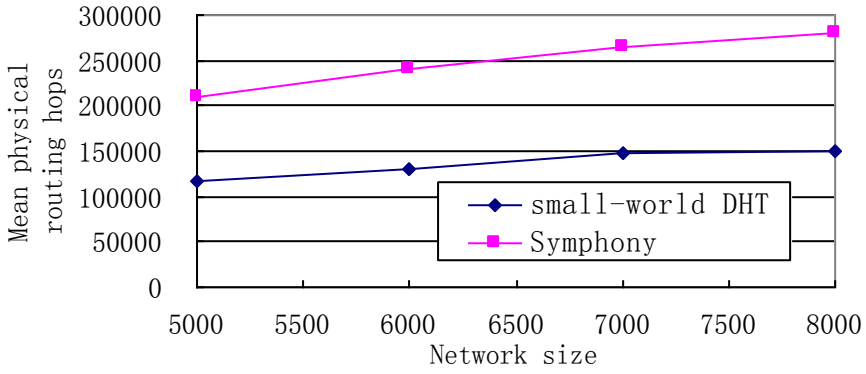

Figure 4. Mean Routing Hops of 200 Lookups in the Small-World DHT and Symphony

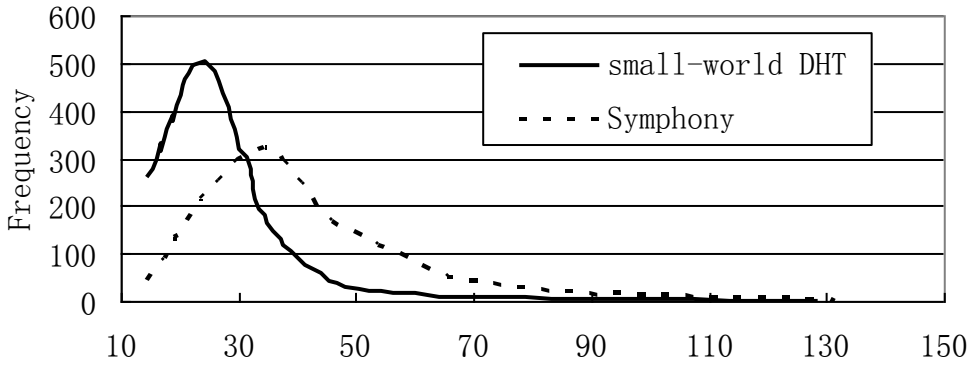

Figure 5. Frequency Distribution of Routing Stretch over 1,000 Lookup Operations in the Small-World DHT and Symphony with 212 Nodes

We use the lookup stretch as the main metric to evaluate the locality performance of both the two networks. Fig.5 plots the frequency of the routing stretch over 1,000 lookup operations in the small-world DHT and Symphony, where the network size is 2^{12} nodes. For a fair comparison, we did not exploit any heuristic to improve the locality performance of both the networks in the experiment. We can see in Fig.5 that the routing stretch of the small-world DHT has a smaller mean and a smaller variance than that of Symphony, which implicates that the underlying geometry of the small-world DHT, a 2-torus, accomplishes better locality performance than a ring- like geometry.

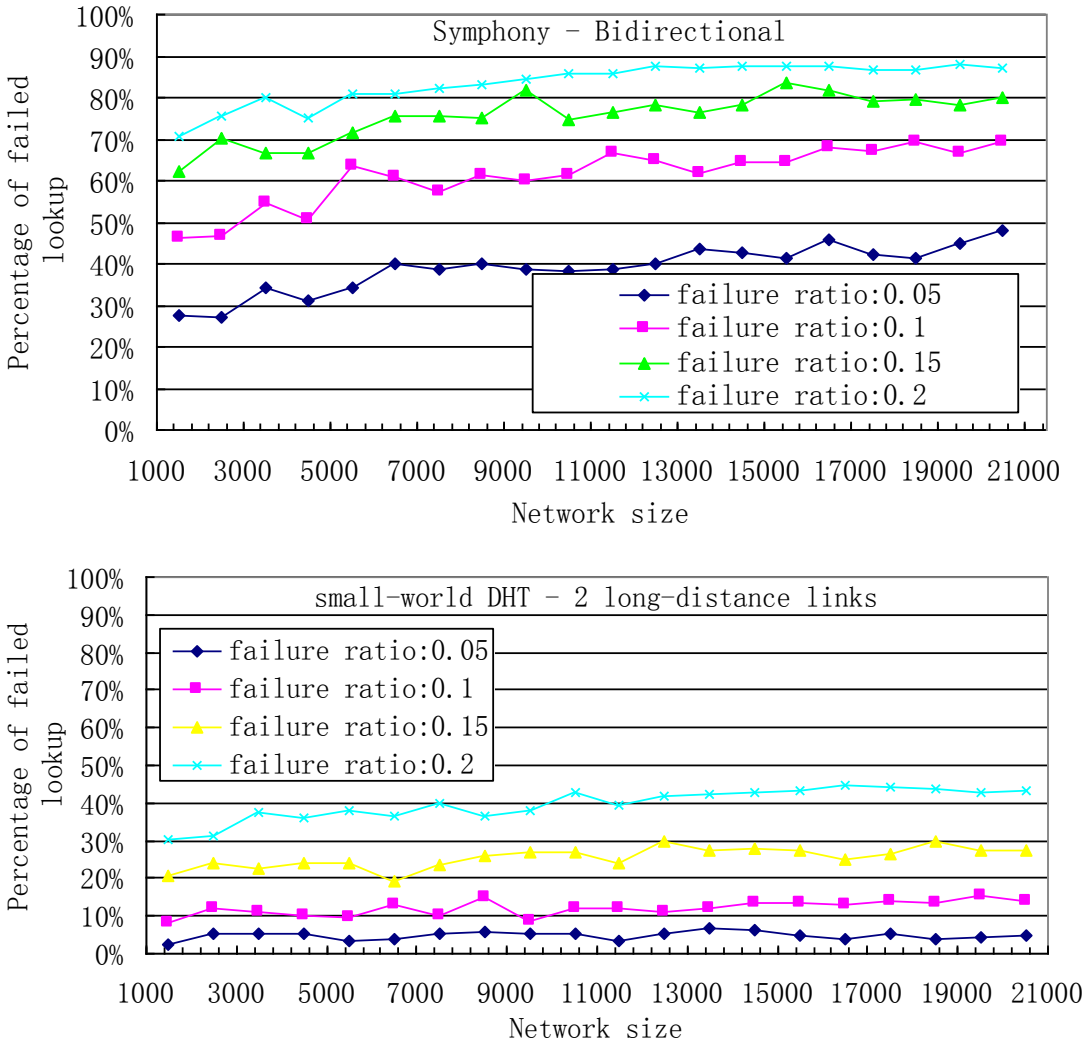

Figure 6. Percentage of Lookup Failures Experienced by Small-World DHT and Symphony

To compare the routing performance of the two networks in the presence of failure, we measure the fraction of failed lookup given a possibility of link failure. The two networks are both equipped with 2 long-distance links in the experiment. Fig.6 illustrates the fraction of failed lookup in each network scale with different failure ratios of links. The proportion of failed lookup in the small-world DHT is obviously lower than that in Symphony, which implicates that our scheme provides greater flexibility since there are more other alternatives for the next hop when the determined next recipient is down.

7. Conclusion

Our scheme requires knowledge of the current network size in order to construct the long-distance links. To estimate the network size, we demonstrated two estimation mechanisms, the flooding approach and the partition tree approach. Both of them are extended from Kleinberg's work, which derives the network size from the number of nodes in a sample area. The former measures the size of the sample area by flooding the request over a restricted search zone and collecting the answers; the latter maintains a distributed binary partition tree and measures the size of the sample area by counting the

number of leaves in the corresponding branch. We analyzed many aspects of their performance and found the partition tree approach was more efficient and balanced than the flooding

Our scheme is guaranteed to recover gracefully from a failure on the assumption that CAN's recovery mechanism functions properly in the presence of a mutable environment. However, further work is needed to verify the correctness of CAN's recovery mechanism.

References

[1] H. C. Hsiao, C. W. Chang. A symmetric load balancing algorithm with performance guarantees for distributed hash tables., IEEE Transactions on Computers, 62(4), 2013, 662-675.
[2] P. Felber, P. Kropf, E. Schiller. Survey on load balancing in peer-to-peer distributed hash tables. Communications Surveys & Tutorials, 16(1), 2014, 473-492
[3] Y. Malkov, A. Ponomarenko, A. Logvinov. Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems, 45(6), 2014, 61-68
[4] W. Yan-liang. Research on Small World Model in P2P Networks. Advances in Information Sciences & Service Sciences, 4(2), 2012, 65-78
[5] S. Jahid, S. Nilizadeh, P. Mittal. DECENT: A decentralized architecture for enforcing privacy in online social networks, 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops),326-332, 2012, Lugano
[6] D. Korzun, A. Gurtov. Survey on hierarchical routing schemes in "flat" distributed hash tables, Peer-to-Peer Networking and Applications, 4(4), 2011, 346-375
[7] C. Harvesf, D. M. Blough. Replica placement for route diversity in tree-based routing distributed hash tables. IEEE Transactions on Dependable and Secure Computing, 8(3), 2011, 419-433
[8] T. Tiendrebeogo, D. Ahmat, D. Magoni. Reliable and scalable distributed hash tables harnessing hyperbolic coordinate, 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), 1-6, 2012, Istanbul
[9] X. Bonnaire. Fixed Interval Nodes Estimation: An accurate and low cost algorithm to estimate the number of nodes in Distributed Hash Tables. Information Sciences, 21(8), 2013, 165-181
[10] G. Fersi, W. Louati, M. B. Jemaa. Distributed Hash table-based routing and data management in wireless sensor networks: a survey. Wireless networks, 19(2), 2013, 219-236
[11] L.Wang, G.Y.Hu, X.L.Sun. The Optimization of EIGRP Algorithm Based on Particle Swarm and Qos Constraint, Journal of Harbin University of Science and Technology, 17(4), 2012, 88-91.
[12] H.W.Wu, R.W.Zhang, H.T.Wang. (k， l) -Anonymity for Social Networks Publication Against Composite Attacks, Journal of Harbin University of Science and Technology, 18(3), 2013, 47-53.

