RChord: An Enhanced Chord System Resilient to Routing Attacks

Dong Xuan, Sriram Chellappan and Muralidhar Krishnamoorthy
Department of Computer and Information Science
The Ohio State University Columbus, OH 43210
E-mail: {xuan, chellapp, krishnmu @ cis.ohio-state.edu.}

***Abstract*— In this paper, we propose a variant of Chord system that is resilient to routing attacks. By routing attacks, we mean the attacks which detour the looking up messages, aiming to disrupt the performance of data look-up systems by increasing the path length of queries. Chord routes messages uni-directionally and has no bi-directional edges. While its performance in the absence of routing attacks is acceptable, it degrades dramatically under routing attacks. Following this observation, we introduce the concept of reverse edges to the Chord system. We named the new Chord system as RChord. We propose several deterministic and randomized algorithms to construct reverse edges. We design a routing algorithm for the new system, which is simple, efficient and backward compatible with the original system. We then analyze the performance of the RChord under routing attacks. We find that its performance is significantly improved in terms of average path length, even by adding very few reverse edges.**

I. Introduction

A Peer-to-Peer (P2P) networked system is one in which several individuals participate in the construction of an independent network. Such a network might be centralized or decentralized, meaning such a network might be constructed with or without a central authority. Such a system performs application level routing on top of IP routing. Based on the organization, a P2P network can be classified as structured and unstructured. Structured P2P systems are those in which nodes organize themselves in an orderly fashion while unstructured P2P systems are those in which nodes organize themselves randomly. Structured P2P systems boast an efficient lookup mechanism by means of Distributed Hash Tables (DHTs) while most of the unstructured P2P systems use broadcast search technique.

The literature on most of the newly developed P2P architectures places more emphasis on the construction, operation and maintenance of the P2P systems in the presence of benign and trustworthy participating nodes. Operation in the presence of malicious nodes is not considered. This is reasonable if the participating nodes are within an isolated network. But in the case of the Internet, it is not. The authenticity of nodes can be administered by a central authorization authority as recommended by Pastry [1]. However, this would undermine some of the basic features of P2P systems. There are many situations in which it is not desirable to constrain the membership of a P2P system. The system must be able to operate even in the presence of malicious nodes.

As a first step is systematically studying the performance of structured P2P system in the presence of malicious nodes, we choose to study the resilience of Chord. By resilience, we mean the ability of the system to maintain performance levels in the presence of compromised nodes (our performance metric is average path length of queries). Our focus in this paper is to study the resilience of Chord, which is a popular structured P2P system [3]. Chord has some salient features which distinguish it from other well-known P2P systems. Chord employs a uni-directional routing mechanism. It is substantially less complicated and handles concurrent node joins and failures well. Data lookups are efficient and available data is guaranteed to be found. In fact one of Chord's better features is; with high probability it resolves data look-ups with a path length of $O\ log(N)$ nodes for an N node system[3]. In proving this property however, the authors do not consider the effect of malicious forwarding. Since Chord was designed for operation in Internet scales, this need not be the case and the path length will be highly impacted in the presence of malicious nodes. In this paper, we study and propose an enhancement of resilience to routing attacks in the current Chord system by focusing on the average path length taken by queries and analyze how it is impacted in the presence of malicious nodes. We study the impact of uni-directional routing on resilience. Based on our findings, we propose to improve resilience by introducing the concept of reverse edges to the Chord system. We name the new Chord system as RChord. We propose two classes of reverse edge adding algorithms: one is deterministic which comprises of the Mirror, Uniform and Local-Remote combination algorithms, and the other is random which is the Local-Remote combination with Randomization algorithm. Due to the existence of reverse edges in the enhanced Chord system, the routing behavior is different from that of the original Chord system. We design a routing algorithm for the new system, which is simple, efficient and backward compatible with the original system. We then analyze the performance of the new Chord system under routing attacks. We find its performance is significantly improved even by adding very few reverse edges. With the *RChord* system, we can achieve performance efficiency greater than 50% under intensive attack conditions. We also find that different reverse edge construction algorithms have different impacts on the performance and they vary as the intensity of attack changes as described later in this paper.

Our paper is organized as follows. In Section 2, we briefly describe Chord and compare it with well known P2P systems and introduce the impact of malicious nodes. In Section 3, we introduce the system and describe its semantics. In Section 4, we analyze the *RChord* system and compare its resilience with

respect to the original Chord system. Section 5 provides the reader with an overview of related work and Section 6 gives concluding remarks.

II. Background and Motivation

In this section, we provide a very brief introduction to Chord and its routing mechanism for the sake of clarity and ease of understanding. To compare with other similar systems, we also give a brief description of other popular structured peer-to-peer systems. We then proceed to describe our motivation in doing this study.

A. Chord

Chord [3] uses a single dimensional circular key space and the node responsible for the key (analogous to data) is the node whose identifier equals or most closely follows the key (numerically); that node is called the key's successor. Each node in Chord maintains two sets of neighbors. Each node has a successor list that has nodes that immediately follow it in the key space. The neighbor list of node n, also called as the finger table is constructed with nodes which are at distances in powers of 2, *i.e.*, the nodes at distances $(n + 2^{i-1}) \mod 2^m$, where $1 \leq i \leq \log N$ in an N-node system form the entries of the finger table as shown in Figure 1. Each node maintains up to m entries in its finger table, where m is the number of bits in the node / key identifier space and the node / key identifiers are generated using hash functions.

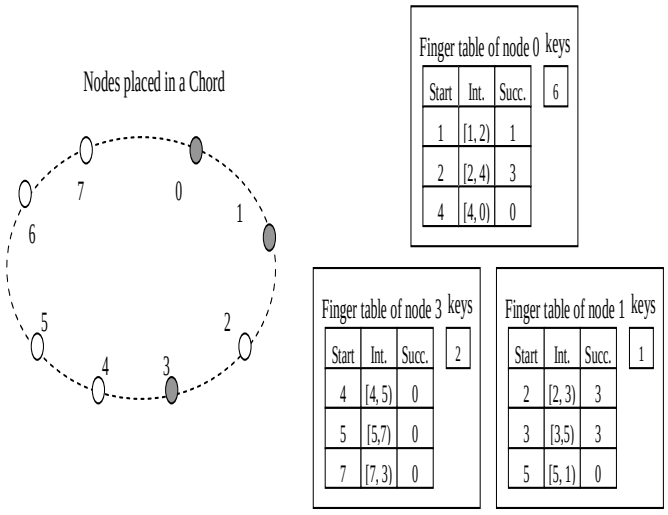


Fig. 1. A simple Chord network and finger tables of the nodes.

To illustrate the routing semantics of Chord, consider Figure 1. The nodes in the Chord system are '0','1' and '3'. Let us consider keys '1','2' and '6'. Thus from the definition above, node '0' is the successor of key '6', node '1' is the successor of key '1' and node '3' is the successor of key '2'. We also show the corresponding finger table for each node. In order to do a search for a key 'k', a node 'n' where the query is currently located searches its finger table to find the node 'j' that immediately precedes 'k' and asks the node 'j' to do the search. By repeating this process, node 'n' ultimately finds the successor of key 'k'. To illustrate this is in Figure 1 , if node '3' wants to search for key '1', then it checks its finger table to find the closest node that immediately precedes key '1'. From the interval (Int) column in the finger table, that node is '0'. Thus node '3' contacts node '0'. Node '0' checks its finger table and finds that its successor is '1' and hence node '1' must contain the required key. This information is then fed back to node '3'. Thus routing is done towards and not past the key.

With the above technique routing technique, with high probability data look-ups are resolved with a path length of $O\ log(N)$ [3]. In the event of failures, the Chord protocol runs a stabilization algorithm and uses its successor list to recover from such failures [3]. One special feature in this routing process that is different from other structured P2P systems is that routing is uni-directional in Chord, In the above example, routing is done in the $clock\ wise$ direction. It is this feature which we will be prominently studying in this paper.

B. Other Structured Peer-to-Peer Systems

a) CAN: Sylvia et. al [2] proposed the Content Addressable Networks as a distributed infrastructure that provides hash table like functionality on Internet-like scales. It models the participating peers as zones in a d-dimensional toroidal space. Each node is associated with a hyper-cubal region of this key space and its neighbors are the nodes, which are associated with the adjoining hybercubes. Routing consists of forwarding to a neighbor that is closer to the key (in the toroidal space). For a d-dimensional space partitioned into N equal zones, the average routing path length is $\frac{d}{4}N^{\frac{1}{d}}$ hops and individual nodes maintain $2d$ neighbors. In CAN systems, multiple paths exist between two points in the space and so even if one or more of a node's neighbors were to crash, a node can automatically route along the next best available path.

b) Pastry: In Pastry [1], nodes are responsible for keys that are the closest numerically (with the keyspace considered as a circle similar to Chord). The neighbors consist of a $LeafSet\ L$, which is the set of L closest nodes (half larger, half smaller). Correct, not necessarily efficient, routing will be achieved with this leaf set. To achieve more efficient routing, Pastry has another set of neighbors spread in the key space. Routing consists of forwarding the query to the neighboring node that has the longest shared prefix with key (and in the case of ties, to the node with identifier numerically closest to the key). In Pastry, in the event of node failures, it uses the repairment algorithm discussed in [1].

c) Tapestry : Tapestry [4] is an overlay location and routing infrastructure that provides location independent routing of messages directly to the closest copy of an object or service using only point-to-point links and without centralized resources. The nodes maintain the routing table called as $neighbormaps$. It's routing along the nodes follow the longest prefix routing similar to the one in the CIDR IP address allocation architecture. The Tapestry location mechanism is almost similar to the Plaxton location scheme [5] but has more semantic flexibility.

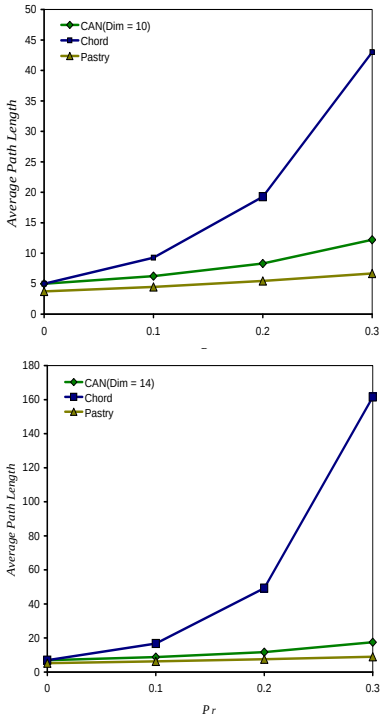
C. Our Motivation

Our motivation to study the resilience of the Chord system is due to a combination of the simplicity and clarity in its design coupled with its vulnerability to attacks. Chord offers a lot of desirable features with minimal complexity. Chord is more robust to partial node failures and node join / leave process is simpler. Also Chord requires minimal maintenance overhead in such cases. The system's performance gracefully degrades in situations where there is a lack of sufficient information for routing. However, in the presence of malicious nodes Chord becomes quite fragile. Our focus in this paper is the resilience of the Chord system to attacks aiming to increase the average path length of queries which is a fundamental performance metric. The average path length is the average number of hops taken by a query to locate a desired key in the P2P system.

A malicious node can cause the malfunctioning of a P2P system in many ways. It is very sophisticated to completely characterize all the effects of a malicious node. However one of the most critical effects of a malicious node is increasing the path length of queries. There are many types of attacks that can increase the path length taken by queries like incorrect routing, not adhering to the protocol rules, co-ordinated attacks by nodes that could make the query toggle back and forth and make the path length theoretically infinite. In this paper, we analyze a simple, albeit very effective attack model, where a malicious node forwards the query to a node that is farthest from the destination than itself. Among the possible attack models, this model is one that can potentially do maximum damage to the system. Throughout this paper this will be the type of attack performed by a malicious node. This attack model can also be applied to study resilience of CAN, Pastry and other such P2P systems. Although our focus in this paper is on the average path length under our proposed attack model, we are in the process of designing more complicated attack models that test Chord's and other structured P2P system's resilience to metrics apart from just the average path length.

In Figure 2 we compare the average path lengths of Chord, CAN and Pastry under our attack model. The system size, (number of nodes) is set as $N = 1024$ and $N = 16384$. Here P_r on the X-axis is the probability that a node in the Chord system performs such an attack. Thus, when a node receives a query to forward; with probability P_r it will behave as a malicious node and will forward the query to a node farthest from the intended destination than itself. Here, we assume all nodes have same P_r, although in reality, this is not the case (P_r is different for different nodes). However we argue that the performance of such a system is highly representative, since the structured P2P systems we study are symmetric, and since we focus on the average performance. The Protocols are simulated in recursive fashion similar to what is described in the original Chord algorithm in [3].

From Figure 2, we can clearly see the impact of direction in the routing of Chord and its resilience. For CAN, the dimension is set as 10 and 14 respectively for the two system sizes analyzed. It can be seen that even though Chord performs well under normal conditions, its performance degrades dramatically in both cases under hostile conditions. Thus Chord



a. $N = 1024$ b. $N = 16K$

Fig. 2. Impact of Direction: Performance evaluation under super malicious routing.

is more susceptible to hostile attacks as compared to CAN and Pastry. This is primarily because of the uni-directional routing mechanism in Chord. CAN and Pastry support bi-directional routing. Hence even if a query overshoots the destination, a benign node will be able to correctly route it by backtracking to the destination in the case of CAN and Pastry, but it is not possible in Chord.

We believe that Chord's resilience can be greatly improved by incorporating bi-directional routing. In the following section, we propose to enhance Chord to make it support bi-directional routing by systematically adding reverse edges and then analyze the performance of the *RChord* system.

III. Enhancing Chord with Reverse Edges

A. Overview

The concept of unidirectional routing in Chord can be misused during a routing attack. An alternative to unidirectional routing would be to have routing take place both in the forward (like a normal Chord node) as well as in the reverse direction depending upon the proximity to the destination. Thereby, even if the request overshoots (crosses) the destination due to some attack, the receptor node can always return the request to the correct node in the reverse direction thus minimizing the path length of the query from that point and thereby making the system more resilient to such attacks. To do that, we face the following two issues:

- How to add the reverse edges to the routing table?
- How to do routing with reverse edges?

In the following two subsections, we will address these two issues in detail.

B. Adding Reverse Edges

One simple approach in adding reverse edges is to include the nodes which are the *mirror nodes* for the nodes present in the routing table of a normal Chord node. Obviously, more reverse edges will lead to better performance, however, it would introduce more overhead and redundancy among the edges. We believe that significant performance improvement can be achieved only by adding a few reverse edges. In this paper, we focus on adding a constant number of reverse edges and propose several approaches to add them. We broadly classify them as deterministic and randomized algorithms.

1) A Deterministic Algorithm to Add Reverse Edges: With this approach, the nodes that the reverse edges point to are determined by some system parameters, for example, the system size. The way the current Chord system adds edges is deterministic. That is, each node has a fixed set of neighbors located at predetermined distances relative to the node. This enables simpler routing procedure and little overhead of computation. Based on this fact, we design several deterministic approaches to add reverse edges to Chord. Before we proceed with our discussion of the algorithm, we will define the notation of reverse neighbors to simplify our description: *A node is said to be the reverse neighbor of node i if the reverse edge from node i points to that node.*

a) The Mirror Algorithm (M): : A simple algorithm to add reverse edges is a straight extension of the Chord's algorithm. With this algorithm, the reverse edges of a given node are mirrors of the fingers (the clock-wise edges in the original Chord system) of that node in the system. This algorithm is formally described in Figure 3. The inputs to the algorithm are the system size, N (number of peers in the system) and the number of reverse edges R [1]. The algorithm determines the reverse neighbors for a given node as follows: for node i, its reverse neighbors r_k are $(i-2^k) \bmod 2^m$, where $k = 0, \ldots, R-1$ and $m = \log N$. We call this algorithm as the *Mirror* (M) algorithm. Figure 4(i) illustrates this simple algorithm. In this figure, the system size N is 256 and the number of reverse edges R is 7 [2]. Hence in such a system, any node i, has the nodes $(i-1) \bmod N$, $(i-2) \bmod N$ ..... , $(i-64) \bmod N$ as its reverse neighbors. For the sake of clarity, the nodes very close to the node i are not shown in the figure.

b) The Uniform Algorithm (U): The *Mirror* algorithm can be easily extended. Note that in Chord (assume the system size is N), for one node i, nodes with id $(i+2^k) \bmod 2^m$ ($k = 0, 1, 2, \ldots, \log N-1$) are selected as node i's finger neighbors. In other words, each node has $\log N$ finger neighbors, which are distributed *uniformly* in the space of $\log$ scale. The benefit of this approach is that the average path length is $O \ \log N$ [3]. In our situation, the number of reverse edges is a constant number, say R. We can divide uniformly the space of $\log$ into R sections, rather than into $\log N$ sections. The algorithm is formally described in Figure 3 and is as follows: for one node i, its reverse neighbors r_k are $(i-2^k) \bmod 2^m$, where $k = \lfloor \frac{(p+1)m}{R+1} \rfloor$, $p = 0, \ldots, R-1$. We call this algorithm as the *Uniform* (U) algorithm for the reason explained above. Figure 4(ii) illustrates this algorithm. For node i, its two reverse neighbors are $(i-4) \bmod N$ and $(i-32) \bmod N$.

c) The Local-Remote Combination Algorithm (L-R):: The final approach we propose in the deterministic section is the *local-remote combination* algorithm. This algorithm attempts to follow the idea of cooperation between local edges (L)and remote edges (R). From the observations made in the above section, it is intuitive to see that this kind of cooperation can improve the performance considerably. We add some edges close to the current node, while add others at a remote distance. They are chosen alternatively. For example, if 3 reverse edges are introduced, the combination would be 2 local ones and 1 remote one. If four reverse edges were introduced, then it would be 2 local ones and 2 remote ones. Figure 3 gives a formal description of this algorithm: for node i, its reverse neighbors r_k are $(i-2^k) \bmod 2^m$, where $k = (m-1-\lfloor \frac{p}{2} \rfloor)(p \bmod 2) + \lfloor \frac{p}{2} \rfloor (1-r \bmod 2)$, $p = 0, \ldots, R-1$ and $r = 0, \ldots, R-1$. In Figure 4(iii), we illustrate the simplest case of two reverse edges with one closer to the node and one farther from the node (in the reverse direction). To node i, its local reverse neighbor is $i-1$, and its remote reverse neighbor is $i - 2^{(\log N - 1)}$.

Input: the system size N ($m = \log N$), the number of reverse edges R, the current node id i
Output: the reverse neighbors r_k ($k = 0, \ldots, R-1$)

Mirror Algorithm:
$r_k = (i-2^k) \bmod 2^m$, where $k = 0, \ldots, R-1$.
Uniform Algorithm
$r_k = (i-2^k) \bmod 2^m$, where $k = \lfloor \frac{(p+1)m}{R+1} \rfloor$,
$p = 0, \ldots, R-1$.
Local-Remote Combination Algorithm
$r_k = (i-2^k) \bmod 2^m$, where $k = (m-1-\lfloor \frac{p}{2} \rfloor)$
$(p \bmod 2) + \lfloor \frac{p}{2} \rfloor (1-p \bmod 2)$,
$p = 0, \ldots, R-1$.
Local-Remote Random Algorithm
$r_k = (i-2^k) \bmod 2^m$, when k is even and less than R, and the probability of a node j to be r_k (where k is odd and less than R) is as follows:
$Prob(j) = (i-j) \bmod 2^m / \sum_{q=1}^{n/2-1} q \bmod 2^m$.

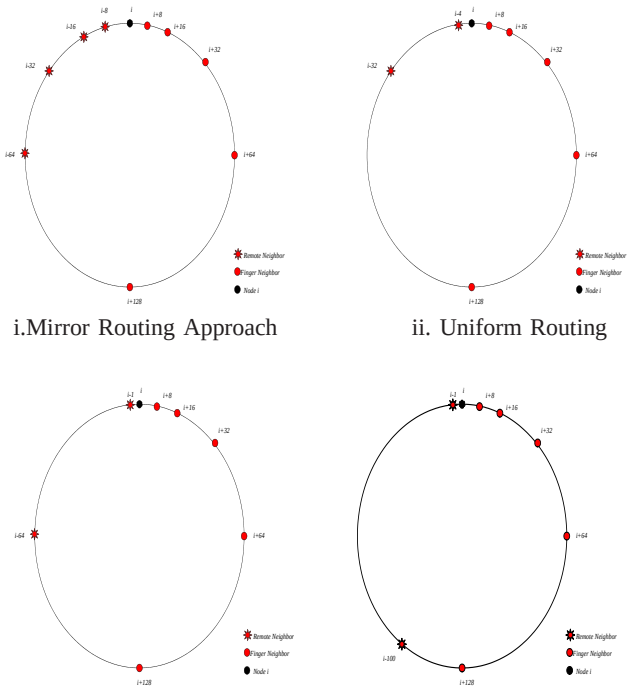
Fig. 3. Algorithms for Adding Reverse Edges for the Chord System

In the next section, we will extend the local-remote combination algorithm by introducing the concept of randomization in adding remote edges.

2) A Randomized Algorithm to Add Reverse Edges: In this section, we will present an approach to add some reverse edges to nodes which are randomly selected. The basic motivations are as follows: (1) Randomly selected reverse edges will make it difficult even for a highly destructive attack to cause extensive damage. (2) An efficient way to add edges can get more performance benefits.

[1]Note that all other algorithms in Figure 3 have the same inputs. To avoid redundancy, we will not describe these parameters when describing the other algorithms.

[2]Note that, all other figures in Figure 4 have the same size of system and the same number of reverse edge numbers. We will not describe these parameters in the other examples.



i.Mirror Routing Approach ii. Uniform Routing

iii. Local-Remote Combination Approach iv. Local Deterministic -Remote Random Combination Approach

Fig. 4. Reverse Edge Adding Algorithms

a) The Local-remote combination algorithm with randomization (L-R-Random):: With this algorithm, the local reverse neighbors are still selected following the same way as algorithm *L-R*, however, the remote reverse neighbors are selected in random fashion. Specifically, the probability of one node to be selected as the remote reverse neighbor is proportional to the distance between the current node and that node. With this algorithm, the node which is far away from the current node has higher probability to be chosen as a remote node. In this sense, the algorithm is similar to the above deterministic local-remote combination algorithm. However, some nodes which are closer to the current node also have some probability to be selected and so we expect the performance to be different from the deterministic one. The formal description of the algorithm is given in Figure 3. We call this algorithm as *L-R-Random* algorithm. Figure 4(iv) illustrates this random algorithm. For node i, its local edge will point to $(i-1)$, while its remote edge may point any node j from node i to node $(i-2^{\log N})$, following the probability: $Prob(j) = (i-j) \bmod 2^m / \sum_{q=1}^{n/2-1} q \bmod 2^m$.

While introducing the reverse edges, either in the deterministic or in the random approaches, the *RChord* follows the same protocol as Chord in discovering and maintaining those reverse edges. For discovering the reverse edges, the location of the remote edges is determined by the algorithm used. For maintaining reverse edges, when a node leaves the system, the immediate successor in the reverse direction, takes up the responsibility of that node, just as the normal Chord does in the deterministic case and in the random case, the node chooses the replacement using the algorithm discussed above.

C. Routing

We need to extend Chord's routing algorithm to consider the reverse edges. Chord's routing algorithm is greedy. Upon receiving a request, a peer will pick a node among its neighbors which is closest to the destination in the clock-wise direction. The algorithm is simple and robust. Our extended algorithm should also have these merits. Also it is important that our algorithm is backward-compatible with the original Chord. That is, if there is no reverse edge, our new algorithm should be equivalent to the original one.

A simple extension is picking the neighbor among the forward and reverse neighbors which is closest to the destination in any one of the two directions. For example, in Figure 5, an enhanced Chord system with 256 nodes using 2 reverse edges is present. We set the source and the destination to be 70 and 0 respectively. Assume the L-R algorithm be used in adding reverse edges. According to the above simple algorithm, node 70's remote reverse neighbor, i.e. node 6 will be selected because it is the closest to destination 0 among all node 70's forward and reverse neighbors. While the algorithm is simple, also compatible with the original algorithm, it is not efficient. The basic reason for that is the asymmetry between the clock-wise edges and the reverse edges. We can use the same example in Figure 5 to illustrate the inefficiency of such an extension. In this example, we find that for node 70, even though its reverse neighbor node 6 is closer to the destination than its forward neighbor node 198, routing through node 6 will take more hops (7 hops) to reach the destination than through node 198 (6 hops).

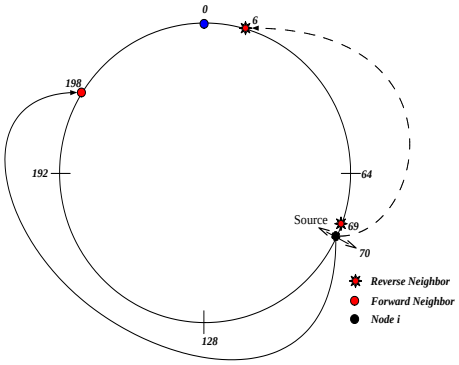


Fig. 5. Erroneous Routing

From the above example, we conclude that our extension should consider the asymmetry between the forward edges and the reverse edges. We need to do some estimation on the routing capabilities of forward edges and reverse edges. Also the forward and reverse edges may work together to achieve good performance. Our routing algorithm should consider this kind of scenario. However the cooperation between the two types of edges will be very comprehensive. Considering all the possible cooperation schemes will lead the algorithm to be complicated, and hence not robust in dynamic P2P systems. Care must be taken to make a good balance between complexity and efficiency. With the above consideration in mind, we design the routing algorithm in Figure 6 for the

enhanced Chord system. The algorithm can work for the system with all different reverse edge adding algorithms.

Inputs : current node i, destination j
Outputs : $NextHop$: the next hop for routing and
$HopNum$: the number of hops taken from node i to node j via $Next\_Hop$

1. Find the forward neighbors closest to the destination in the forward and reverse direction, F_f and F_r
2. Find the reverse neighbors closest to the destination in the reverse and forward direction, say R_r and R_f
3. Compute the number of hops taken by each of the four candidates
   H_{ff} = uni_forward_routing(F_f);
   H_{rf} = uni_reverse_routing(F_r);
   H_{fr} = uni_forward_routing(R_f);
   H_{rr} = uni_reverse_routing(R_r);
   $HopNum$ = min(H_{ff}, H_{rf},H_{fr}, H_{rr}) + 1;
   $NextHop$ = node_with_min_hops(F_f, H_{ff}, F_r, H_{rf},R_f, H_{fr}, R_r, H_{rr});
   return $NextHop$ and $HopNum$

Fig. 6. The routing algorithm for the enhanced Chord system

In this algorithm, functions $uni\_forward\_routing()$ and $uni\_reverse\_routing()$ are used to estimate the average number of hops from one node to another node assuming only one-direction edges be used in the routing. The pseudo codes of two uni-direction routing functions are presented in Appendix A.

The routing algorithm returns two values: one is the next hop that node i should route the query to; another is the number of hops that the query needs to route from node i node to destination j. We define the latter as the *enhanced-Chord-distance* from node i to node j. Let's take the example in Figure 5 to illustrate this algorithm. We still set the source and the destination to be 70 and 0. To node 70, its F_f is node 198, and its R_r is node 6. It has no F_r and R_f. H_{ff} =5, and H_{rr} = 6, hence $NextHop$ is node 198, and $HopNum$ is 6. With this algorithm, node 198 rather than node 6 will be selected to be the next hop for node 70 to node 0. The *enhanced-Chord-distance* of node 70 to node 0 is 6.

Note that this routing algorithm has the following features: (1) It is compatible to the original Chord routing algorithm in the sense that: If there is no reverse edge, there will be no F_r, R_r and R_f. Hence, there is only one candidate node F_f which can be selected as the next hop, which we know is the node that is selected by the original Chord routing algorithm. (2) It considers the cooperation between the forward and reverse edges. With this algorithm, four neighbors F_f, F_r, R_r and R_f are considered to be the next hop. If node F_r is finally selected, the query first is routed to F_r in the forward direction, and may be routed close to the destination along the reverse direction. The case is similar to node R_f if it is selected. However, this algorithm limits to one simple scenario: each routing has only one chance to change its direction; (3) The difference in the routing capacity of the forward and reverse edges are considered by calling the function of $uni\_forward\_routing()$ and $uni\_reverse\_routing()$ during routing decision.

IV. Analysis of the RChord System

In this section we will analyze the impact of introducing reverse edges to the Chord system. Particularly we will analyze the impact of the different reverse edge adding algorithms, i.e. Mirror (M), Uniform (U), Local-Remote combination (L-R) and Local-Remote combination with Randomization (L-R-Random) on the Chord system and draw some important conclusions which would act as guidelines in designing a robust Chord system.

Figure 7 (i-iv) shows the comparison in performance (the average path length) for different system sizes under varying intensity of attacks (different P_r). The reverse edge size is fixed at 2 for all the cases. From these figures, we can make the following observations:

- Compared with the original Chord system, the resilience of the *RChord* system has improved significantly, particularly for the the U, L-R and L-R-Random algorithms. This becomes more prominent as the intensity of attack (P_r) increases. For example, in the case of system with 1024 nodes, when P_r = 0, the performance efficiency of U, L-R and L-R-Random algorithms are 11%, 12% and 13% respectively. But when the intensity of attack is 0.3, the performance efficiencies are increasing drastically to 44%, 53% and 43% respectively, ie., almost three times the ideal condition. It is easy to believe that the effect of having a constant number of reverse edges will decrease with the increase in system size. However our results show that there is no such significant decrease in the degree of performance. This can be explained based on the fact that, as the system size increases, the reverse nodes though constant in number are at different positions relative to the node having the reverse edges to those nodes. The position of the reverse edges also influences the performance of the algorithms.
- Comparing the performance of systems using different algorithms, we find that the overall performance of the Mirror algorithm is poor, and ones for other three algorithms vary depending on the different value of P_r:
  - when $P_r < 0.1$, the performance of the system with different algorithms are similar to each other.
  - when $0.1 < P_r <= 0.3$, the performance of the system using U is better than one using L-R-Random, which outperforms one using L-R. This observation can be explained by the fact that with the Uniform algorithm, the whole system is well covered by the edges (including the reverse edges and the forward edges). Under mild attack conditions, the regular routing mechanism still functions. The coverage of system by the edges will help achieve good performance.
  - when $P_r > 0.3$, the performance of the system using L-R is better than one using L-R-Random, which outperforms one using U. It can be explained by the fact that when P_r is very large, the chance to use

the regular routing to reach the destination is very small. The query is misdirected among the nodes in the system due to the super routing attack. With the L-R algorithm, the remote reverse edge is very useful to redirect the query to the node closer to the destination, and the local reverse edge can be a good supplement to the remote reverse edge to send the query to the destination.

V. Related Work

There is a lot of current ongoing research work in the area of improving security features of P2P systems. We will discuss some of them which are closely related to ours. Emil Sit and Robert Morris discussed the security considerations for P2P systems in [12]. They propose a set of design principles which can make the P2P systems, stronger and more secure against attacks. The PIPE (P2P Information Preservation and Exchange) network [13] designed by B.F.Cooper et. al attempts to protect data stored in a P2P environment from malicious nodes using replication mechanisms. Other P2P systems like LOCKSS [14] and Archival Intermemory [15] also use replication mechanisms similar to PIPE to prevent the denial of service attack to a certain extent. Unlike these replication based systems, PoET [16] provide security by setting access rights to data available in the P2P system. Certain systems like CFS [17] uses node ID's which has a hash of the node's IP address in them thereby preventing malicious misdirection. A clone of Chord known as Achord [18] was proposed to provide censorship resistance. It's design was inspired by the architecture of Freenet [19] which is an unstructured P2P system.

All the above work in P2P systems are interesting and try to provide security in the framework of data replication and access control mechanisms imposed on the data in the system. Our focus in this paper is to study the impact of basic system design features on the resilience. We are able to point out how malicious nodes can take advantage of otherwise good semantics of the system and we also show how to significantly improve performance with minimal changes to the system design.

VI. Final Remarks

In this paper we studied the resilience of the average path length of Chord to routing attacks. and pointed out the importance of bi-directional routing.

We then proposed an enhanced Chord system along with several reverse edge adding approaches and a routing algorithm. Our enhanced system is simple and efficient in the sense that it outperforms significantly the original Chord system in terms of the average path length under routing attacks, even with very few reverse edges. Another good feature of our system is that is backward compatible with the original Chord system.

There are several directions to extend our study potentially: (1) It will be interesting to theoretically prove the features of our reverse edge adding algorithms and hence derive an algorithm, that will give optimum performance. (2) We can extend our study to other Peer-to Peer systems, and to other types of attacks, for example dropping and cooperative routing attacks. (3) We can include other metrics apart from just the average path length to study.

References

[1] A. Rowstron and P. Druschel, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content addressable network.* In Proc. ACM SIGCOMM, 2001.

[3] I. Stoica, R. Morris, D. Karger, M. F.Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications.* In Proc. ACM SIGCOMM 2001, August 2001.

[4] Y. Zhao, J. Kubiatowicz, and A. Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2000. 16.

[5] C. G. Plaxton, R. Rajaraman, and A. W. Richa, *Accessing nearby copies of replicated objects in a distributed environment.* In Proceedings of ACM SPAA. ACM, June 1997.

[6] Sylvia Ratnasamy, Scott Shenker and Ion Stoica, *Routing Algorithms for DHTs: Some Open Questions.* In proc. of IPTPS, March,2002.

[7] Andrew.S.Tenenbaum,Maarten Van Steen *Distributed Systems: Principles and Paradigms*, Prentice Hall, 1st Ed.

[8] A. Ravindran, Don T. Phillips, James J. Solberg, *Operations Research: Principles and Practice, 2nd Edition*, John Wiley & Sons.

[9] Randolph Nelson, *Probability, Stochastic Processes, and Queueing Theory: the Mathematics of Computer Performance Modelling*, Springer-Verlag.

[10] Dong Xuan, Muralidhar Krishnamoorthy and Shengquan Wang, *Enhanced Chord: Technical Report*, Technical Report, Department of Computer and Information Science, The Ohio-State University,Columbus, http://www.cis.ohio-state.edu/ xuan.

[11] R. Motwani and P. Raghavan, *Randomized Algorithms* Cambridge University Press, Cambridge, 1995.

[12] E. Sit and R.Morris *Security Considerations for Peer-to-Peer Distributed Hash Tables* In Proc. of IPTPS 2002.

[13] B.Cooper,M.Bawa, N.Daswani,H.Garcia-Molina. *Protecting the PIPE from malicious peers* Technical Report, Stanford.2002.

[14] V.Reich and D.Rosenthal *Lockss(Lots of copies Keep stuff safe)* Preservation 2000, Nov.2000.

[15] Y.Chen et al. *A prototype implementation of archival intermemory* In Proc. of the ACM Conf. on Digital Libraries, 1999.

[16] S.Payette and C.Lagoze. *Policy-carrying,policy-enforcing digital objects* In Proc. European Conf. on Digital Libraries (ECDL), 2000.

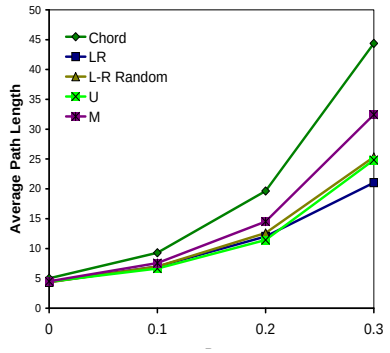
[17] F.Dabek et al. *Wide-area cooperative storage with CFS* In Proc. of ACM Symposium on Operating Systems Principles. 2001

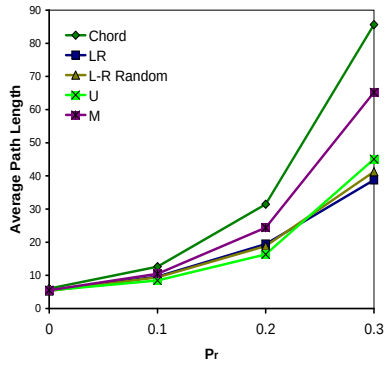
[18] S.Hazel and B.Wiley. *Achord: A variant of the Chord Lookup Service for Use in Censorship Resistant Peer-to-Peer Publishing Systems* In Proc. of IPTPS 2002.

[19] I.Clarke et al. *Freenet: A Distributed Anonymous Information Storage and Retrieval System* In Designing Privacy Enhancing Technologies:International workshop on Design Issues in Anonymity and Unobservability, 2001.

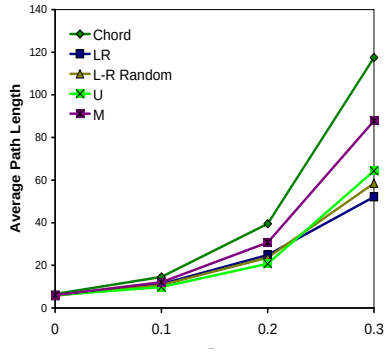
Appendix A: Uni-directional Routing Functions

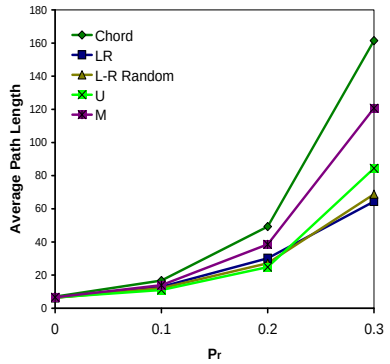
This section provides the pseudo code for the unidirectional forward and reverse routing functions. These functions determine hops on the assumption that the routing is to take place only in that direction. It is relatively easy to do that for the deterministic edges. However it is harder to do so for the randomized edges. This can also be regarded as one of merits of this approach, since it is difficult for the attacker to figure out a super attack strategy. For the randomized edges, the algorithm is based on a simple assumption that the node assumes that every other node would have a random reverse edge at the same distance as its random reverse edge.





(i)$N = 1024$ (ii)$N = 4096$





(iii) $N = 8192$ (iv)$N = 16384$

Fig. 7. Performance Evaluation of the Enhanced Chord System under Super Routing Attack.

Input: Source node i, destination node j, the system size N, and the number of reverse edges R

Output: the next node and the number of hops need to be taken from node i to node j

uni_forward_routing()

{

$dist = (i - j) \text{mod} N$;

/* compute how many factors of 2 add to dist */

for $i = logN - 1$ to 0

$hops1 = dist/(power(2, i)$;

$dist = dist - power(2, i) * hops1$;

$hops = hops + hops1$;

next i;

return $hops$;

}

uni_reverse_routing()

{

/* $STEPSIZE[\,]$: Array of location of reverse neighbors determined by the reverse edge adding algorithms*/

$dist = (i - j) \text{mod} N$;

for $i = 1$ to R

$hops1 = dist/(power(2, STEPSIZE[i]))$;

$dist = dist - power(2, STEPSIZE[i]) * hops1$;

$hops = hops + hops1$;

next i;

return $hops$;

}

Fig. 8. Unidirection Routing Functions