HELSINKI
INSTITUTE FOR
INFORMATION
TECHNOLOGY

CR-Chord: Improving Lookup Availability in the Presence of Malicious DHT Nodes

Dmitry Korzun, Boris Nechaev, Andrei Gurtov

HIIT Technical Reports 2008-2

Dmitry Korzun
Helsinki Institute for Information Technology HIIT
PO Box 9800, FIN-02015 TKK, Finland
`dkorzun@hiit.fi`

Boris Nechaev
Helsinki Institute for Information Technology HIIT
PO Box 9800, FIN-02015 TKK, Finland
`boris.nechaev@hiit.fi`

Andrei Gurtov
Helsinki Institute for Information Technology HIIT
PO Box 9800, FIN-02015 TKK, Finland
`gurtov@hiit.fi`

Helsinki Institute for Information Technology HIIT
HIIT Technical Reports 2008-2
ISSN 1458-9478 (electronic)

Helsinki Institute for Information Technology HIIT
Tietotekniikan tutkimuslaitos HIIT

Spektri site	*Kumpula site*	*Otaniemi site*
Mailing address:	Mailing address:	Mailing address:
HIIT	HIIT	HIIT
PO Box 9800	PO Box 68	PO Box 5400
FI-02015 TKK	FI-00014 University of Helsinki	FI-02015 TKK
Finland	Finland	Finland
Visiting address:	Visiting address:	Visiting address:
Spektri Business Park	University of Helsinki	Helsinki University of Technology
Pilotti Building	Department of Computer Science	Department of Information and Computer
Science		
Metsänneidonkuja 4	Gustaf Hällströmin katu 2b	Konemiehentie 2
02130 Espoo	00560 Helsinki	02150 Espoo
tel: +358 9 4511	tel: +358 9 1911	tel: +358 9 4511
fax: +358 9 694 9768	fax: +358 9 191 51120	fax: +358 9 451 3277

www.hiit.fi

CR-Chord: Improving Lookup Availability in the Presence of Malicious DHT Nodes

Dmitry Korzun*, Boris Nechaev* and Andrei Gurtov*
*Helsinki Institute for Information Technology HIIT / Helsinki University of Technology TKK
PO Box 9800, FIN-02015 TKK, Finland
E-mails: dkorzun@hiit.fi, boris.nechaev@hiit.fi, gurtov@hiit.fi

Abstract—**Distributed Hash Tables (DHTs) provide a useful key-to-value lookup service for many Internet applications. However, without additional mechanisms DHTs are vulnerable to attacks. In particular, previous research showed that Chord is not well resistant to malicious nodes that joined the DHT. We introduce the cyclic routing algorithm as an extension of Chord (CR-Chord). Using simulations we compare the lookup availability of Chord and CR-Chord. The results suggest that CR-Chord improves the lookup availability on the average by 1.4 times. When the number of malicious nodes is small, such as 5%, CR-Chord has almost twice lower lookup failure rate.**

I. Introduction

Nowadays Distributed Hash Tables (DHT) are a part of many peer-to-peer (P2P) applications in the Internet. To mention a few examples, DHTs are used to track the upload/download ratings in Bittorrent and resolve host identifiers to IP addresses for Host Identity Protocol (HIP) [1]. Each DHT node maintains a routing table of its neighbors containing node identifiers (IDs) and IP addresses. The main service provided by DHTs is routing a lookup query for a certain key to a DHT node that stores the value for that key.

As Internet applications increasingly depend on DHTs to operate, DHT should be resilient to all kinds of attacks. One of the most dangerous scenarios is when adversaries are able to become a part of DHT by joining as regular nodes. Then attackers can corrupt, drop, or misroute lookup messages. We restrict our study with dropped lookups only.

Let *lookup failure rate* be the probability that an arbitrary lookup is dropped. Complementary, *lookup availability* is the probability that a lookup arrives at the destination. Several techniques for improving lookup availability in the presence of malicious nodes were proposed, including iterative routing and lookup progress monitoring [2], [3], self-certifying data [4]–[6], routing failure tests and root verification [6], [7].

We propose cyclic routing (CR) [8] as a way to enhance robustness of existing DHTs in the presence of malicious nodes. In this paper, we focus on integration of cyclic routing with Chord [9], which is one of the first and still most popular DHTs.

In CR-Chord, a node maintains a collection of cycles additionally to its finger table. A cycle is a path that starts from the node to its finger, then runs through the network and returns to the node. Only IDs of cycle nodes, but not IP addresses, are stored. Cycles present global knowledge about good paths in a DHT. If there is a cycle containing a node close to the destination, the message is sent along this cycle. Otherwise, Chord is used to select the next hop.

We implemented CR-Chord as an extension of the MIT Chord simulator. With simulations, we analyzed and compared the Chord and CR-Chord lookup availability. The results suggest that CR-Chord improves the lookup availability by 1.4 times on the average.

The rest of the paper is organized as follows. Section II provides background on DHT security and defines the problem of dropped lookups. Section III describes the CR-Chord algorithm. In Section IV, we define our simulation methodology. Section V explains the simulation results of CR-Chord vs. Chord. In Section VI, we summarize the most important findings and experiences with CR-Chord.

II. Background and Motivation

A. DHT Routing

Consider a DHT consisting of N nodes. Node IDs are assigned from an identifier space with a distance metric. Each node s maintains a routing table T_s of entries (u, IP_u), where u is a neighbor and IP_u is its IP address. Hence, s forwards messages to u via the underlying IP network. A message to a destination node d goes sequentially to nodes whose IDs are progressively closer to d according to the distance metric. If $v \in T_s$ then s can forward a message to v forming the one-hop path $s \to v$. Routing from s to d takes several hops forming a multi-hop path $s \to^+ d$.

According to [8], DHT routing is divided onto *global* and *local* parts. In global routing, a message is delivered close to the destination. In local routing, the destination is at a nearby node. The reasons for division are as follows. (A) Since a node is responsible for the keys closest to its ID, let a lookup message arrive to a node close to the key. (B) Various replication techniques support routing into an area of neighboring nodes. (C) A DHT node knows its neighborhood well keeping close nodes to its routing table when possible. Obviously, global routing is more vulnerable to attacks.

DHT routing is either *iterative* or *recursive*. With iterative routing each node on the lookup path returns the next-hop node v to the querying node. The latter then contacts v to iteratively get closer to the destination. With recursive routing each node forwards lookups directly to the next hop nodes, and the querying node receives a response from the destination. Iterative routing is more secure since a querying node can control the routing progress. Nevertheless, more network resources are consumed, and iterative routing is not possible when a querying node cannot directly contact some nodes on the path, e.g., due to NATs. In this paper, we consider recursive routing only.

The lookup availability depends on the number of alternate paths between a source and destination [3], [10], [11]. The more paths, the more chances to go around malicious nodes. However, a mechanism for finding paths consisting of good nodes is needed. The straightforward approach exploits multi-path routing when nodes *multicast* messages to several neighbors [6], [12]–[14]. It has two disadvantages. First, a lot of duplicate messages are generated; many of them are redundant due to the local selection (non-optimal) of alternate paths. Second, in some DHTs alternate paths converge [10], [13], [15], and bypassing malicious nodes becomes impossible.

B. Chord

Chord [9] uses an identifier circular space of n-bit integers (modulo 2^n), and participating nodes form a ring taking IDs from $[0, 2^n - 1]$. A node forwards messages in clockwise direction. The distance $\rho(u, w)$ is length of the clockwise ring arc between u and w. Key k is assigned to the first node whose ID is equal or follows k clockwise.

A Chord routing table for a node s consists of three types of neighbors: (1) successors, (2) fingers, and (3) predecessors. Successors and predecessors are several closest nodes to s, clockwise and counterclockwise, respectively. Successors are node's short-distance contacts aiming at local routing. The ith finger of s is the node that succeeds s by at least 2^{i-1} on the ring, where $i = 1, 2, \dots, n$. Fingers are node's long-distance contacts aiming at global routing. Predecessors are mainly for routing table maintenance.

In a lookup for a key k, each node finds the closest preceding neighbor v. When v is a finger, each hop at least halves the distance. Eventually, a lookup arrives at the node whose immediate successor is responsible for k.

Basic Chord restricts a finger table with n entries. They are updated systematically. More efficient routing is achieved when a node can keep *additional fingers* having more knowledge about the network [16]. Let n_{af} denote the number of additional fingers. When Chord occasionally discovers a node, it is inserted as an additional finger. Periodically, Chord removes fingers that have not been used.

C. Attacks by Dropping Lookups

In attacks on routing a malicious node drops messages, modifies them, or forwards incorrectly. We consider only the case of dropped lookups. Such a lookup fails to reach the destination, and no response is sent.

Dropped lookups concern routing security as well as fault-tolerance. In recursive routing detection of lookup failures is more difficult than in iterative case. Dropped lookups are harder to reveal compared to incorrect lookups; a querying node has no response, and the lookup validity cannot be checked. In fact, only straightforward timeout techniques detect such failures.

Multicast mitigates dropped lookups. Instead of the only next-hop, a node uses several neighbors to forward a message (several alternate paths). The efficiency, however, depends on the number of disjoint paths m [3]:

$$f^m \le \Pr(\text{lookup failure}) \le \left(1 - (1-f)^l\right)^m , \qquad (1)$$

where f is the fraction of malicious nodes (uniform distribution), l is the number of hops in a lookup.

Eq. (1) is valid for any DHT, but Chord is more sensitive than many others. Its finger selection is restrictive, and the only node that immediately succeeds $s + 2^{i-1}$ may be the ith finger. This is avoided in randomized-Chord [15], [17], where the ith finger is taken randomly from $[s + 2^{i-1}, s + 2^i)$. Moreover, introducing additional fingers allows s to know several neighbors in $[s + 2^{i-1}, s + 2^i)$.

In Chord, any path $s \to^+ d$ goes through p, the d's immediate predecessor [3], [10], defining the so called *shield problem.* Although Chord has alternate paths $s \to^+ p \to d$, they are not disjoint, and p becomes a single point of failure for all lookups to d. Therefore, Chord satisfies Eq. (1) only for $m = 1$, and the lower bound can be refined[1] [10]:

$$1 - (1-f)^2 \le \Pr(\text{lookup failure}) \le 1 - (1-f)^l. \qquad (2)$$

Multicast helps even when there are no disjoint paths. However, the problem is in finding a good path among available. DHT routing works locally, and a node just selects several next hops without much knowledge about the remaining paths. In this paper, we offer a more systematic way that uses the results of previous lookups. Having a path that a successful lookup followed lately, a node uses it for subsequent lookups. This idea can be implemented using the concept of cyclic routing.

D. Cyclic Routing (CR)

We follow [8] where CR was originally proposed. Let a node s send a message to a node d ($s \to^+ d$). Then d replies to s ($d \to^+ s$). The paths form the cycle $s \to^+ d \to^+ s$. Taking intermediate nodes, $c = \left(s \to v_1 \to^+ v_2 \to^+ \cdots \to^+ v_{l-1} \to^+ s\right)$, where $v_1 \in T_s$ (direct IP contact). Nodes v_1, …, v_{l-1} may represent some but not all the nodes visited.

Let each node s maintain a collection of cycles additionally to its routing table T_s, $\mathcal{C}_s = \{c_1, \dots, c_q\}$, where $c_j = \left(s \to v_{j,1} \to^+ \cdots \to^+ v_{j,l-1} \to s\right)$ for $j = 1, \dots, q$. The collection $\mathcal{C}_s$ is a network *cyclic structure* known to s. A node constructs the remaining route using the cyclic structure when appropriate. Otherwise, the underlying DHT is called to find the next-hop node. Algorithm 1 shows the details; it is executed at every node $u \ne d$ of $s \to^+ d$.

A simple reactive strategy is used for constructing cycles. When receiving a successful lookup response, s keeps the cycle if it is efficient (e.g., a low number of hops). The underlying DHT facilities are used more optimally; only good and efficient paths are collected among available ones. Note,

[1]This lower bound is valid for $N \ge 16$.

Algorithm 1 Cyclic routing a message to a node d.

```
Require: Message $p$ arrives at $u \neq d$.
  The node $u$ maintains $T_u$ and $\mathcal{C}_u$.
  Find $c \in \mathcal{C}_u$ such that
     $c = \big(u \to v_1 \to^+ \widetilde{d} \to^+ u\big)$ where $\widetilde{d}$ is close to $d$;
  if $c$ is found then
    Let the next-hop node $v$ be $v_1$;
  else
    Find the next-hop node $v \in T_u$ according to the underlying DHT;
  end if
  Forward $p$ to $v$;
```

however, that the strategy should be considered as a catalyst; it fails when the underlying DHT cannot produce good paths, and another way for constructing cycles is needed.

III. Integration of Cyclic Routing with Chord

Algorithm 1 works on top of any DHT. In this section, we adapt cyclic routing to the Chord DHT. The resulting system is called CR-Chord and follows Algorithm 2.

Each node s maintains a cyclic structure $\mathcal{C}_s$ exploiting regular Chord lookups. Let s initiate a lookup for a key k. The message is delivered to the destination d according to Chord routing. Then d sends an acknowledgment to s. All nodes on the path $s \to^+ d \to^+ s$ form a cycle c.

There are two types of lookup messages in CR-Chord, *primary* and *secondary*. Primary lookups may exploit cycles. Secondary lookups are used for constructing cycles and do not use cycles. Both lookup types allow finding requested documents. A source sends a primary lookup and multicasts secondary lookups with the *multicast degree* m_d. Since at every node cyclic routing is not applied to secondary lookups, cycles reflect good paths available in a regular Chord network.

Although the lookup success is our primary goal, CR-Chord remains efficient by storing only those cycles that satisfy the performance criterion. Let l be the number of hops in c and f_s be the number of fingers at s. Then c is stored when

$$l \leq k_\mathrm{h} \min(f_s, n), \tag{3}$$

where $\min(f, n)$ approximates $\log_2 N$ (the average number of hops in a Chord cyclic path $s \to^+ d \to^+ s$), k_h is a tradeoff parameter between performance and security. In our simulations, we allow $k_\mathrm{h} = 2$ for bypassing malicious nodes.

At each step, a node u should select a next-hop v to forward p. There are three options. (1) A cycle c_p piggybacked at p is used by u for routing. (2) A better cycle is available in $\mathcal{C}_u$. (3) None of cycles in $\mathcal{C}_u \cup \{c_p\}$ is appropriate, and the Chord-provided choice is used.

In Algorithm 2, u calls $BestCycle$ to decide either to use the previously fixed cycle c_p or there is a better cycle in $\mathcal{C}_u$. The criterion is the closeness according to the Chord distance ρ (see $CycleDist$). If a cycle contains a node $\widetilde{d}$ that is closer to k than u, then the cycle is appropriate. Therefore, u searches for the best cycle among appropriate ones. When no appropriate cycle exists, u simply performs regular Chord forwarding. Otherwise, u finds in c_p the finger v_cycle that is closest to k, piggybacks c_p into p, and forwards p.

A cycle piggybacked into a packet does not consume much space. Chord node ID is 160-bit long (20 bytes). In a Chord network of 10^6 nodes, the cycle length does not exceed 20 with high probability ($\log_2 10^6 < 20$), which results in 400 bytes-long cycles. IP packets with a typical Maximum Transmission Unit of 1500 bytes can easily include such a cycle. Number of cycle nodes piggybacked into a packet can be reduced by storing not a full cycle, but a contiguous subset of its nodes. After a lookup traverses all these nodes, according to Algorithm 2 it will follow the regular Chord algorithm until it finds a new cycle or reaches the destination.

Algorithm 2 The CR-Chord pseudocode.

```
Require: A node $u$ receives a lookup packet $p$ for a key $k$.
  Let $(v_1, \dots v_{m_\mathrm{d}})$ be the closest to $k$ fingers in $T_u$
  if $u = s$ then
    Send secondary lookups to $(v_1, \dots v_{m_\mathrm{d}})$ {Multicast}
  end if
  Let $c_p$ be a cycle piggybacked in $p$ (if any)
  $c_p = BestCycle(\mathcal{C}_u \cup \{c_p\}, k)$
  $v_\mathrm{cycle} = NextCycleHop(c_p, k)$
  if $v_\mathrm{cycle} \neq \varnothing$ then
    Piggyback $c_p$ into $p$
    Send $p$ to $v_\mathrm{cycle}$ {Forward along the cycle}
  else
    Send $p$ to $v_1$ {Forward via the underlying Chord}
  end if

$BestCycle(\mathcal{C}, k)$
  Find $c$ in $\mathcal{C}$ such that
     $CycleDist(c, k)$ is minimal and $CycleDist(c, k) < \infty$
  return $c$ if found, and $\varnothing$ otherwise

$NextCycleHop(c, k)$
  Find in $c$ the closest node $v$ to $k$ such that
     $v \in T_u$ and $\rho(v, k) < \rho(u, k)$
  return $v$ if found, and $\varnothing$ otherwise

$CycleDist(c, k)$
  Find in $c$ the closest node $\widetilde{d}$ to $k$ such that $\rho(\widetilde{d}, k) < \rho(u, k)$
  return $\rho(\widetilde{d}, k)$ if found, and $\infty$ otherwise
```

IV. Simulation Methodology

A. Attack model

Let $M = fN$ be the number of malicious nodes, where $0 \leq f < 1$ is the malicious nodes fraction (further for readability we express f as percentage). Then $G = N - M$ is the number of good nodes. Malicious and good nodes are distributed uniformly in the Chord ring.

Malicious nodes attack the network by dropping lookups. Since such a node pretends to be good as much as possible, we assume that it processes correctly all routing maintenance traffic. Otherwise, its malicious activity can be detected easier. To some degree this behavior is similar to a faulty or overloaded node; its malicious but imperceptible activity includes the following.

1) Dropping lookup and acknowledgment messages (no forwarding).
2) Processing data placement requests as a good node but no real data are stored.
3) Ignoring lookup requests for data items for which the node is responsible.

Without loss of generality we also assume that malicious nodes do not send lookup requests.

We assume that the network has reached a stable state where data items are distributed uniformly among nodes. If any data happen to be at a malicious node then the data are lost. There are no more data insertion and deletion. Routing maintenance goes as if all nodes are good.

In this stable state the network provides the lookup service for data by keys. A random good node u queries a lookup for a given key k. Assuming that u selects k uniformly, the lookup failure and success rates characterize the availability of the lookup service.

B. *Simulation scheme*

Our simulation of a Chord network with malicious nodes follows Algorithm 3. We used the MIT original Chord simulator (http://cvs.pdos.csail.mit.edu/cvs/sfsnet/simulator/) for network sizes $N = 1000, 2000, 3000$ and ID space with $n = 24$. Variation of N provides a more adequate analysis of the lookup availability [10].

For every value of N, the fraction of malicious nodes f varies from 5% or 10% up to 50%; the increment step is either 5 or 10. When varying N, parameters D (Step 2) and L_{rate} (Step 4) are proportional to N and N^2, respectively.

Algorithm 3 Simulation steps

1. *Initially a good network.* A network of G good nodes is constructed. Nodes are joining randomly.

2. *Data placement.* D data items are distributed in a good network. In all simulations $D = 100N$.

3. *Introducing malicious nodes.* M malicious nodes join the network at the same time.

4. *Serving lookups.* L requests are performed. For a request a pair (u, k) is selected randomly, where u is a good node and k is the key for a data item stored at step 2. There are two phases, *Stabilization* and *Analysis*, $L = L_{\mathrm{stab}} + L_{\mathrm{rate}}$ requests.

Stabilization (for CR-Chord only). Good nodes initiate L_{stab} requests without multicast. The requests are used for constructing cycles but they are not counted in the lookup availability estimation.

Analysis. Good nodes initiate $L_{\mathrm{rate}} = 0.01N^2$ requests. They are analyzed for the success or failure. In CR-Chord, a request induces one primary lookup and m_{d} secondary lookups. In Scenario 7, churn of both good and malicious nodes is performed during this phase.

The MIT simulator implements the basic Chord [9]; the ith finger of s is the first node in $[s + 2^{i-1}, s + 2^i)$ for $i = 1, 2, \ldots, n$. The randomized-Chord [15], [17] would lead to better results since nodes have more freedom in selecting fingers, and more good paths are available.

We enhanced the MIT simulator with CR-Chord implementation (see Algorithm 2 above). In our version, the size of $\mathcal{C}_s$ is unbounded and no maintenance for cycles is implemented. It does not, however, affect the results much. First, the attack model assumes a stable network. Second, the number of cycles constructed per node is moderate or even small for large M (see Fig. 10, 11 and 12). Note that in practice the maintenance cost for node IDs is less than for fingers since the former (i) does not involve expensive IP address maintenance and (ii) uses piggybacking to regular lookups.

Table I summarizes our simulation scenarios. Each scenario consists of two parts, for Chord and CR-Chord, respectively. Each part takes ten executions of Algorithm 3 for averaging.

C. *Churn model*

To assess the CR-Chord behavior in a dynamic environment we introduce churn. In simulations, nodes join and leave the network at a constant rate R. At the same time good nodes generate lookup requests. The ratio of good and malicious nodes joining network (parameter f) is preserved constant; each join event is accompanied by leave event.

This churn model comprises only dynamics of nodes, not documents. The latter are inserted before churn starts. During churn, documents are moved to appropriate nodes. When a node joins the network it acquires from its successor all the documents it becomes responsible for. When a node, either good or malicious, leaves the network it hands over all its documents to the successor. For simplicity we omit a more realistic model where documents stored at malicious nodes are lost completely.

According to [9] the churn rate R is the number of joins/leaves per second. We vary churn rate from 0.01 to 0.2. Note that in our simulations lookups are generated with interval of 1 second and stabilization of nodes is done every 30 seconds on average. Thus, for boundary values of R each churn event is performed after every 100 and 5 lookups respectively. Such rates are consistent with the previous studies. In [18] a node generates lookup every 0.1 second, while churn events occur at the rate of 0.067 to 8 per second, which corresponds to a churn event happening every 150 to 1.25 lookups. In [9] for the same lookup generation rate as ours, churn rate varies from 0.05 to 0.4, which is equivalent to 20 and 2.5 lookups between two churn events.

D. *Success and failure types*

A request for a key includes a primary lookup and m_{d} secondary lookups. Either of the lookups or both of them can succeed by reaching the responsible node. To find out the share

TABLE I
SUMMARY OF SIMULATION SCENARIOS.

Scenario	N	f	L_{stab}	n_{af}	m_{d}	R
Typical simulation configuration						
1	1000	5%, 10%, …, 50%	0	12	3	0
Requesting data when a responsible node and its immediate predecessor are good						
2	1000	10%, 20%, …, 50%	0	12	3	0
Variation of the network size, N						
3	1000, 2000, 3000	10%, 20%, …, 50%	0	12	3	0
Variation of the number of additional fingers, n_{af}						
4	1000	10%, 20%, …, 50%	0	0, 12, 24, 48	3	0
Variation of the multicast degree, m_{d}						
5	1000	10%, 20%, …, 50%	0	12	1, 2, …, 10	0
Variation of the stabilization period, L_{stab}						
6	1000	10%, 20%, …, 50%	0, 1000, 5000, 10000	12	3	0
Variation of churn rate, R						
7	1000	10%, 20%, …, 50%	0	12	3	0.01, 0.02, 0.05, 0.1, 0.2

of each we compute detailed successes and failures metrics. Table II presents the statistics for Scenario 1.

We distinguish three success cases:

1) Only primary lookup succeeds, all secondary lookups fail (*Primary lookups success*);
2) At least one of secondary lookups succeeds, primary lookup fails (*Multicast lookups success*);
3) Both primary and at least one of secondary lookups succeed (*Joint success*).

The total success rate (*Successful requests*) is the sum of the above values. Note that only one (first reached) successful secondary lookup contributes to the success rate disregarding of the other $m_{\mathrm{d}} - 1$ secondary lookups.

The metrics below show how actively successful primary lookups (*Primary lookups success*) use cycles:

1) A successful primary lookup's cycle was first inserted at the initiator node (*Initiator cycle success*);
2) A successful primary lookup's cycle was first inserted at an intermediate nodes (*Intermediate cycle success*);
3) A successful primary lookup does not contain a cycle (*No cycle success*).

Besides success metrics we estimate primary lookup failure rates (*Primary lookups failure*):

1) The responsible node d is malicious (*Primary lookups failure at responsible node*);
2) The predecessor node of a key k is malicious (*Primary lookups failure at predecessor*);
3) A primary lookup failed at an intermediate node (*Primary lookups failure in the middle*);

There are also two primary lookup failure types that are solely due to churn and are relevant only for Scenario 7:

1) A document cannot be found at the responsible node due to churn (*Primary lookup document churn failures*);
2) A primary lookup failed along the route due to churn (*Primary lookup routing churn failures*).

A primary lookup document churn failure happens when the lookup arrives at the predecessor after the successor leaves the network, but before predecessor is notified about that. Similarly, when a new node joined between the predecessor and former successor, but the Chord stabilization is not completed.

There is no timeout mechanism to retransmit lost packets. It leads to the primary lookup routing churn failures. At each moment a node has a number of requests to forward. If the node leaves the network, all unsent requests are lost. Similarly, a primary lookup can fail along the route due to churn since nodes do not ping fingers before forwarding.

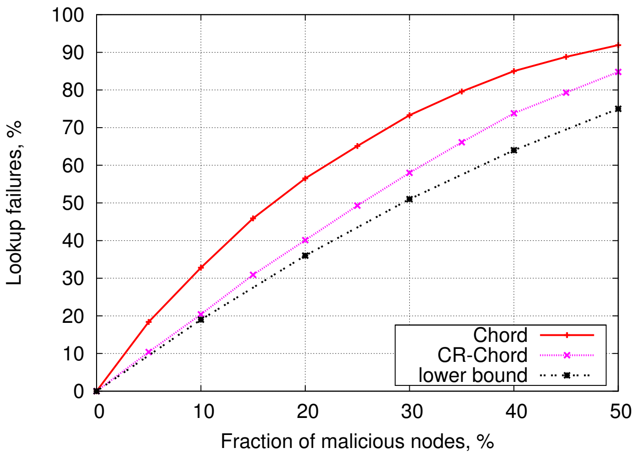

Fig. 1. Lookup failure rate (Scenario 1).

V. Analysis of Lookup Availability

For our simulations, Scenario 1 is typical; Table II presents its key metrics. Other scenarios are focused on particular aspects of lookup availability. The lower bound of Chord lookup failure rate is defined in Eq. (2).

A. Basic facts

Fig. 1 confirms [3], [10] that Chord is not well resistant to presence of malicious nodes. The lookup failure rate exceeds much the lower bound. The peak is for moderate malicious cases ($20\% \leq f \leq 35\%$). For $f = 10\%, 20\%, \ldots, 50\%$ the lookup availability is 32% on average.

TABLE II
LOCATIONS OF LOOKUP FAILURES IN CHORD AND CR-CHORD (SCENARIO 1).

Percent of malicious nodes	10%		20%		30%		40%		50%	
	Chord	CR-Chord	Chord	CR-Chord	Chord	CR-Chord	Chord	CR-Chord	Chord	CR-Chord
Successful requests	6724.9	7958.4	4353.8	5985.4	2673.9	4197.9	1501.1	2617.3	808.2	1518.5
Primary lookups success	0.0	55.3	0.0	88.2	0.0	89.2	0.0	64.8	0.0	39.9
Multicast lookups success	0.0	859.0	0.0	1161.9	0.0	1121.5	0.0	847.8	0.0	534.7
Joint success	0.0	7044.1	0.0	4735.3	0.0	2987.2	0.0	1704.7	0.0	943.9
Primary lookups success	6724.9	7099.4	4353.8	4823.5	2673.9	3076.4	1501.1	1769.5	808.2	983.8
Initiator cycle success	0.0	3287.2	0.0	1912.5	0.0	963.8	0.0	373.3	0.0	137.1
Intermediate cycle success	0.0	1105.8	0.0	626.6	0.0	316.4	0.0	127.9	0.0	45.1
No cycle success	0.0	2706.4	0.0	2284.4	0.0	1796.2	0.0	1268.3	0.0	801.6
Primary lookups failure	3275.1	2900.6	5646.2	5176.5	7326.1	6923.6	8498.9	8230.5	9191.8	9016.2
- at responsible node	743.2	789.4	1037.8	1144.2	1130.6	1300.9	1017.4	1219.0	834.8	1010.0
- at predecessor	814.1	862.5	1375.6	1521.2	1587.0	1817.7	1670.2	1956.0	1601.6	1916.0
- in the middle	1717.8	1248.7	3232.8	2511.1	4608.5	3805.0	5811.3	5055.5	6755.4	6090.2

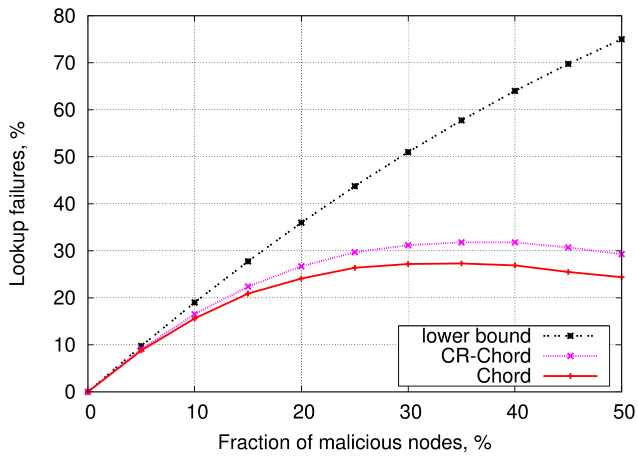

Fig. 2. Lookup failure rate at responsible and predecessor nodes (Scenario 1).

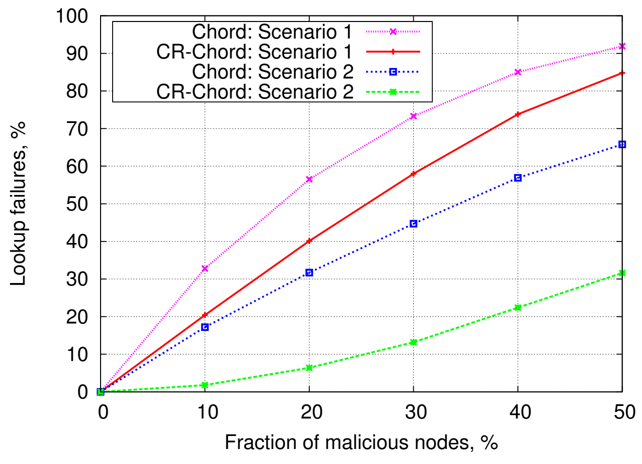

Fig. 3. Lookup failure rate when no failures at responsible node and predecessor (Scenario 2).

The CR-Chord lookup failure rate is lower. For $f = 5\%$ it is 1.77 times less than for Chord (10.4% and 18.4% respectively). In contrast to Chord, the difference with the lower bound monotonically grows when increasing f. The lookup availability of CR-Chord is 45% on average. Comparing with basic Chord, CR-Chord has 7–16% better absolute lookup availability; the best result is for moderate malicious cases. On average CR-Chord performs 1.4 times better; the absolute effect is 13%. For 50% of malicious nodes lookup availability of Chord and CR-Chord is 8.1% and 15.2% respectively; thus for big values of f lookup availability is almost doubled.

A lookup fails either at a responsible node, at its immediate predecessor, or in the middle of the route. The lower bound reflects the first two failures. However, a request can fail earlier even having malicious responsible node or immediate predecessor, and the actual failure rate in Fig. 2 is below the lower bound.

According to the widespread opinion the shield problem is most essential for Chord lookup availability. The last rows in Table II show, however, that failures in the middle constitute a comparable part. It means that a solution to the shield problem does not necessarily lead to high lookup availability.

In fact, the shield problem relates to local routing but failures in the middle affect global routing. CR-Chord has more failures at responsible nodes and immediate predecessors than Chord (average absolute difference is 3%) because of reducing failures in the middle (average absolute difference is 7%). Consequently, CR-Chord optimizes global routing.

Scenario 2 provides further clarification. Any request satisfies two extra requirements: 1) it is for data stored at a good node and 2) its immediate predecessor is also good. The scenario focuses on global routing since all lookups can fail only in the middle. Fig. 3 shows the results; Scenario 1 is for comparison. Chord suffers much from failures in the middle. In contrast, CR-Chord reduces such failures up to 3–4 times.

Most failures in Chord happen at the few first hops, see Fig. 4 (all failed lookups are 100%; cases for $f \neq 20\%$ are similar). CR-Chord routes better around malicious nodes, and less failures happen at the beginning. It is yet another confirmation of optimized global routing in CR-Chord.

B. Variations

Previous research showed that lookup availability varies for different network sizes [10]. To validate how this affects CR-Chord we ran scenario 3 simulations that estimate lookup availability for different values of N (Fig. 5). Clearly, growing

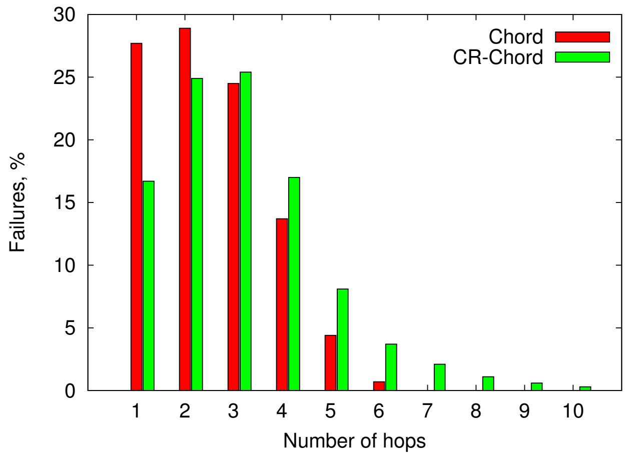

Fig. 4. Distribution of lookup failures according to hop length (Scenario 1, $f = 20\%$).

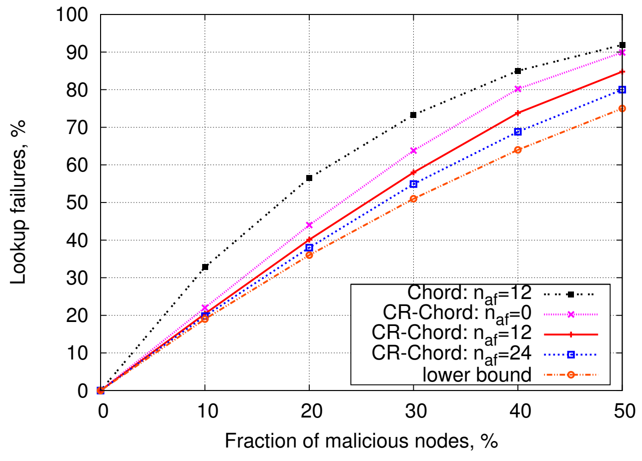

Fig. 6. Lookup failure rate for different numbers of additional fingers (Scenario 4).

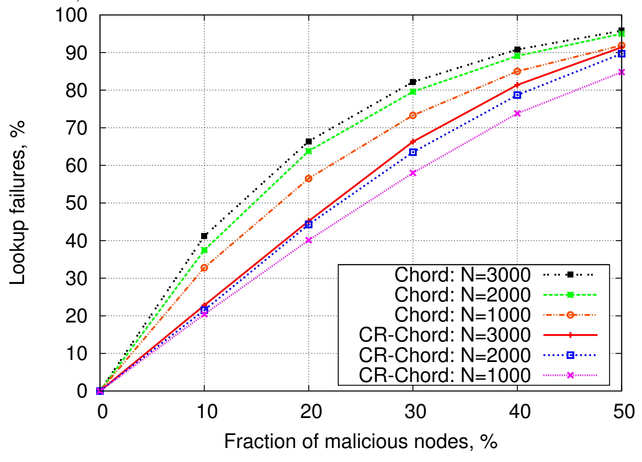

Fig. 5. CR-Chord lookup failure rate for different network sizes (Scenario 3).

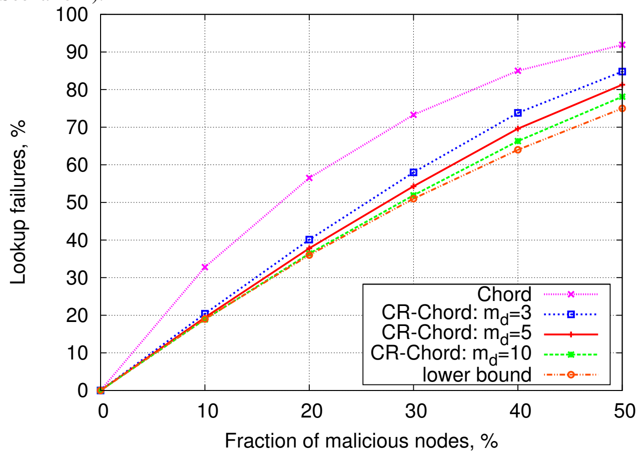

Fig. 7. Lookup failure rate for different multicast degrees (Scenario 5).

N increases the number of hops in lookups, and the lookup failure rate becomes higher (see also Eq. (1)). However since CR-Chord improves global routing, the effect of higher number of hops is mitigated by use of cycles. This is proved by lookup availability ratio. For $N = 3000$ the lookup availability of Chord and CR-Chord is 25% and 39% respectively. Thus CR-Chord performs 1.56 times better than Chord, while for $N = 1000$ if does only 1.4 times better. The failure rate seems to grow sublinearly, and the reason is $O(\log N)$-paths.

Scenario 4 focuses on the effect of additional fingers (Fig. 6). Increasing n_{af} converges the failure rate to the lower bound. However, introducing too many fingers does not help much; doubling n_{af} improves only by a constant.

Scenario 5 aims at improving the lookup availability by multicast (Fig. 7). Basic Chord does not support multicast ($m_{\mathrm{d}} = 0$). In CR-Chord, increasing m_{d} converges the failure rate to the lower bound (as with additional fingers). Even small multicast degree ($m_{\mathrm{d}} = 3$) reduces essentially the failure rate. Further increasing of m_{d} does not affect much.

Scenario 6 considers the effect of stabilization period (for CR-Chord only). Intuitively, with larger L_{stab} more cycles are collected before the actual measurement, and better values are estimated for the lookup availability. Actually, this dependence is negligible since the average difference in lookup availability is about 0.1% of L_{rate}. Hence $L_{\mathrm{stab}} = 0$ is used in other simulation scenarios. Moreover, it is an indicator of good dynamic properties of CR-Chord since only a short stabilization period is required to adopt cycles to the current network state.

Scenario 7 analyses dynamic properties of CR-Chord under churn. Fig. 8 shows that Chord is weakly resistant to the high churn rate $R = 0.2$. Churn-related losses along the route are much more prevalent than inability to find target document. For example, for $R = 0.2$ and $f = 10\%$ Chord has only 35 document failures but 2077 along-the-route loss failures for the total of 10000 requests.

CR-Chord helps to mitigate the negative effect of churn. The average difference in number of lookup failures among $R = 0$ and $R = 0.2$ is lower for CR-Chord than for Chord. CR-Chord has less loss along the route. This is yet another confirmation that CR-Chord improves global routing. For moderate churn rates CR-Chord has lookup availability very close to that shown by simulations without churn, which proves its good dynamic properties.

C. *Analysis of cycles*

Cyclic routing allows transitions when a node changes the cycle obtained previously. Fig. 9 shows the cycle usage in

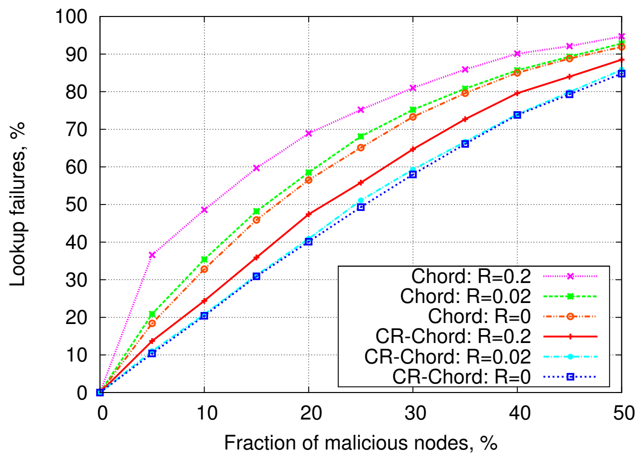

Fig. 8. Lookup failure rate for different churn rates (Scenario 7).

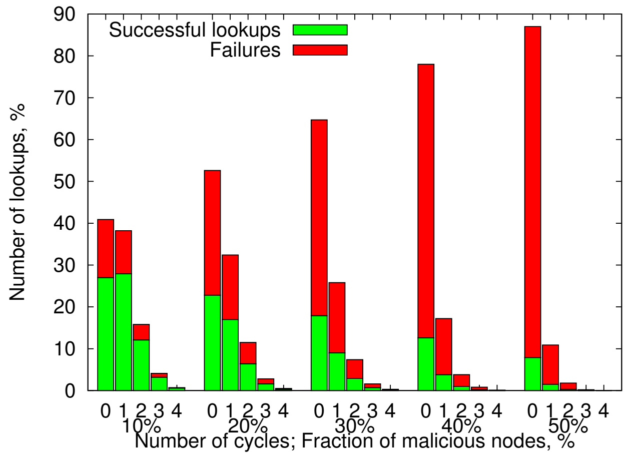

Fig. 9. Distribution of lookups among the number of cycles used (Scenario 1).

lookups. Many successful lookups follow one or two cycles only; the number of lookups with more cycles is essentially less. It validates that cycle routing is stable to changing a cycle, and previously fixed cycle is likely to be preserved.

There are lookups without cycles, when nodes cannot find appropriate cycles because of the small number of cycles at a node for large f, see Figs. 10–12. Additional fingers (Fig. 10) and multicast (Fig. 11) do not help much in constructing cycles. Additional fingers increase the number of constructed cycles almost uniformly for all f while the multicast efficiency decreases for large f. Churn has negative effect since some cycles become incorrect (Fig. 12).

CR-Chord lookup hops are either along cycles or use regular Chord. Fig. 13 shows the cycle hops share in successful lookups. For instance, for $f = 10\%$ about 16% of lookups use cycles for $21\% \ldots 30\%$ of hops. There are no successful lookups with $1\% \ldots 10\%$ cycle share of hops.

Many successful lookups do not use cycles. In particular, they share 45%, 58% and 70% of all successful lookups for $f = 10\%, 20\%, 30\%$ respectively (not shown in Fig. 13). Nevertheless, when there are enough cycles available the essential part of a successful lookup is due to cycles. Thus

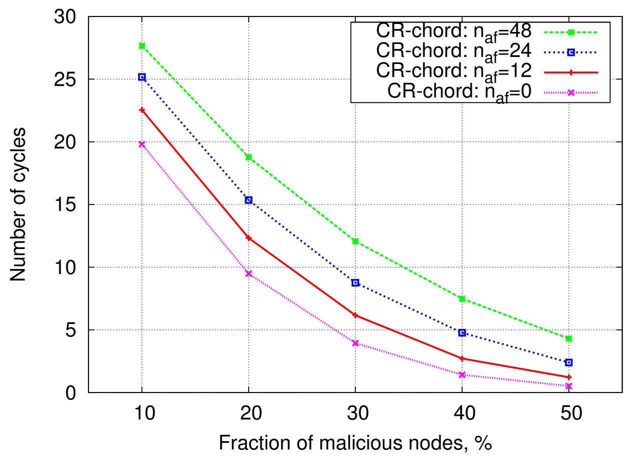

Fig. 10. Number of cycles per node for different numbers of additional fingers (Scenario 4).

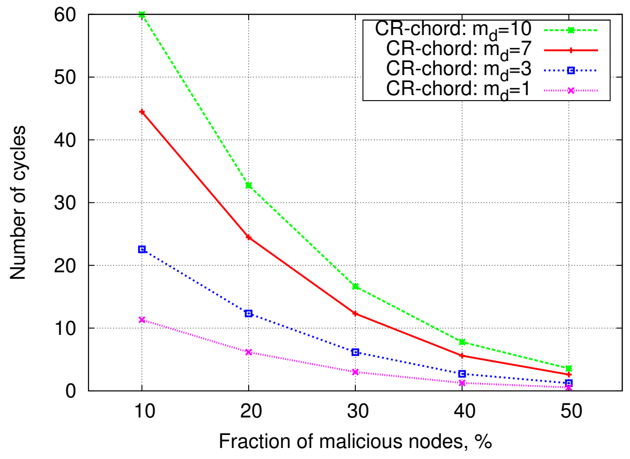

Fig. 11. Number of cycles per node for varying multicast degree (Scenario 5).

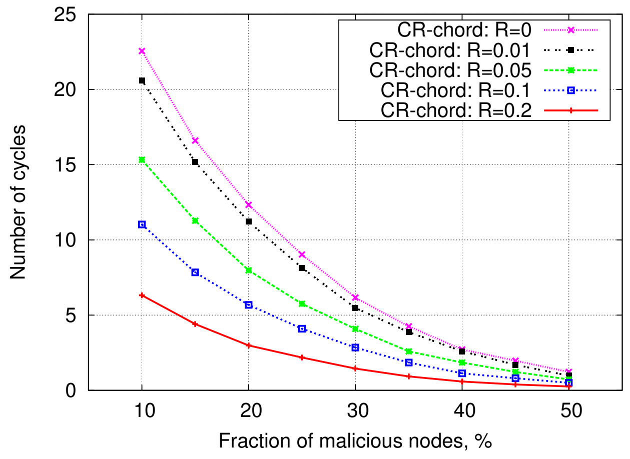

Fig. 12. Number of cycles per node for varying churn rate (Scenario 7).

the problem is in cycle construction, not in cyclic routing.

Besides the hop count, the lookup length can be measured with the Chord distance. The simulation (Scenario 1) identifies two extreme cases. Either a successful lookup does not use cycles, or $91\% \ldots 100\%$ of its distance is along cycles (41%, 30% and 19% for $f = 10\%, 20\%, 30\%$ respectively).

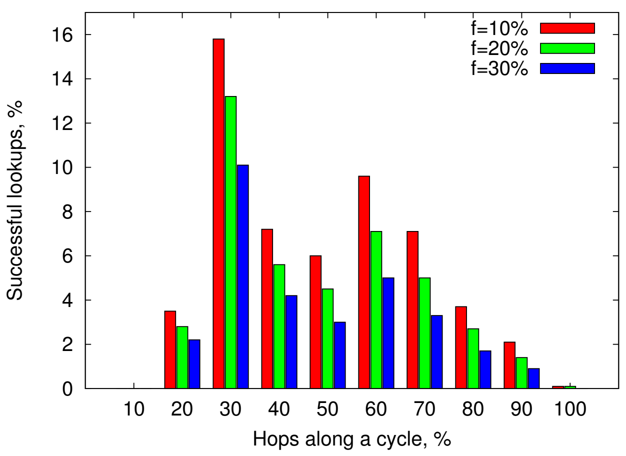

Fig. 13. Distribution of successful lookups along the cycle share in hops (Scenario 1).

VI. Conclusion

We have compared the resilience of CR-Chord and Chord in the presence of malicious nodes. The lookup availability of CR-Chord is on the average 1.4 times higher compared to Chord, and up to two times higher for big number of malicious nodes. The absolute gain in availability of 7%–16% is reasonable compared to existing enhancements for Chord. For instance, zig-zag routing [19] improves the lookup availability at most by 6% for $10\% \leq f \leq 50\%$ [10].

The improvement depends on the number of cycles stored by nodes. For large f and R, cycles are constructed insufficiently, since our simple construction method is based on Chord lookups. It aims merely at more efficient usage of Chord routing facilities. As a result, there are only few cycles per node and the difference between Chord and CR-Chord is small.

A core problem of Chord (as of other DHTs) is that lookups can fail although good paths do exist. In CR-Chord, good paths are stored for future use, but an efficient method for cycle construction is needed. This is a topic of our further research.

Weak local routing in Chord does not allow better routing than dictated by the lower bound in Eq. (2). In CR-Chord, however, global routing is optimized. Combining CR-Chord with a mechanism for secure local routing would improve the lookup availability. Secure local routing seems an easier problem compared to global routing. Node's neighborhood is limited and its exhaustive secure maintenance is possible.

Our simulations suggest that CR-Chord can not only mitigate malicious activity, but also compensate routing churn losses caused by unconcerned nodes routing policy.

We also confirm the idea of combining additional fingers and cyclic routing [8]. Some nodes have resources to maintain more fingers than in basic Chord. When inserting a finger is not possible due to IP addressing restrictions, CR helps in keeping additional information about the network.

References

[1] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Experimental host identity protocol (hip)," IETF RFC 5201, 2008. [Online]. Available: http://tools.ietf.org/html/rfc5201

[2] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 261–269.

[3] M. Srivatsa and L. Liu, "Vulnerabilities and security threats in structured overlay networks: A quantitative analysis," in *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 252–261.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2001, pp. 202–215.

[5] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 188–201, 2001.

[6] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Proc. of the 5th USENIX Symposium on Operating System Design and Implementation (OSDI 2002)*. Boston, MA, USA: ACM Press, Dec. 2002, pp. 299–314.

[7] P. Wang, N. Hopper, I. Osipkiv, and Y. Kim, "Myrmic: Secure and robust DHT routing," Digital Technology Center, University of Minnesota, DTC Research Report 2006/20, Nov. 2006.

[8] D. Korzun, B. Nechaev, and A. Gurtov, "Cyclic routing: Generalizing lookahead in peer-to-peer networks," May 2009, to appear in 7th ACS/IEEE International Conference on Computer Systems and Applications.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," *IEEE/ACM Trans. on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[10] J. Seedorf and C. Muus, "Availability for structured overlay networks: Considerations for simulation and a new bound on lookup success," in *The 12th Nordic Workshop on Secure IT Systems*, Oct. 2007, pp. 23–34.

[11] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, "Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience," *IEEE/ACM Trans. on Networking*, vol. 13, no. 5, pp. 1107–1120, 2005.

[12] K. Hildrum and J. Kubiatowicz, "Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks," in *proceedings of the 17th International Symposium on Distributed Computing (DISC)*, 2003, pp. 321–336.

[13] B. Y. Zhao, L. Huang, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, "Exploiting routing redundancy via structured peer-to-peer overlays," in *Proc. of The 11th IEEE International Conference on Network Protocols (ICNP '03)*, Atlanta, GA, Oct. 2003, pp. 246–257. [Online]. Available: citeseer.ist.psu.edu/article/zhao03exploiting.html

[14] B. Leong, B. Liskov, and E. Demaine, "Epichord: parallelizing the chord lookup algorithm with reactive routing state management," in *Proceedings of the 12th International Conference on Networks 2004 (ICON 2004), Singapore*, Nov. 2004, pp. 270–276.

[15] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proc. of ACM SIGCOMM'03*. New York, NY, USA: ACM Press, Aug. 2003, pp. 381–394.

[16] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek, "Bandwidth-efficient management of DHT routing tables," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005, pp. 99–114.

[17] H. Zhang, A. Goel, and R. Govindan, "Incrementally improving lookup latency in distributed hash table systems," in *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2003, pp. 114–125.

[18] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. of the USENIX Annual Technical Conference*, June 2004.

[19] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson, "Sybil-resistant dht routing," in *10th European Symposium On Research In Computer Security*, 2005, pp. 305–318.

This technical report describes the CR-Chord algorithm and simulations showing that CR-Chord improves the lookup availability in the presence of malicious DHT nodes.

HIIT Technical Reports 2008-2

ISSN 1458-9478 (electronic)