R^5N: Randomized Recursive Routing for Restricted-Route Networks

Nathan S. Evans
Technische Universität München
Munich, Germany
Email: evans@net.in.tum.de

Christian Grothoff
Technische Universität München
Munich, Germany
Email: grothoff@net.in.tum.de

***Abstract*—This paper describes a new secure DHT routing algorithm for open, decentralized P2P networks operating in a restricted-route environment with malicious participants. We have implemented our routing algorithm and have evaluated its performance under various topologies and in the presence of malicious peers. For small-world topologies, our algorithm provides significantly better performance when compared to existing methods.**

I. Introduction

Distributed Hash Tables (DHTs) [13], [11] are a key data structure for the construction of completely decentralized applications. DHTs are important because they generally provide a robust and efficient means to distribute the storage and retrieval of key-value pairs.

In recent years, DHT designs have become increasingly efficient and robust under churn [9], [12], [14], [18] and Sybil attacks [10], [16], [19]. Other research has addressed implementation concerns, such as optimizing network performance. In practice, modern DHTs restrict participation to so-called super-nodes, excluding peers with limited connectivity from direct participation. The primary reason for this is that virtually all previous DHT routing algorithms (with the notable exception of Freenet [15]) are based on the fundamental assumption of universal connectivity between all participating nodes (or rely on unstable NAT traversal).

This assumption means that modern DHTs cannot function properly in networks with limited connectivity (mobile, ad-hoc wireless, sensor, friend-to-friend, etc.). Following [15], we refer to these networks where peers are not free to directly connect to arbitrary other peers (and therefore route in the DHT) as *restricted-route* networks. We need to distinguish between the network topology created by a peer-to-peer overlay and the underlying network infrastructure, so we use the term restricted-route underlay topology to describe the resultant restrictions imposed on the overlay routing algorithm.

This paper introduces a new randomized DHT routing algorithm, R^5N, which enables our DHT to operate effectively over restricted-route networks and also increases security and resilience to various attacks compared to existing algorithms. R^5N only assumes that the topology is connected and, in particular, does not require or use a coordinate system for organizing peers. A primary goal of R^5N is providing an open network where users can join or leave at any time without approval by a certificate authority or other trusted entity.

The R^5N design itself is relatively simple, essentially combining a random walk with recursive Kademlia-style [11] routing. Our design also includes topology augmentation using a combination of distance-vector and onion-routing, a novel replication strategy and an API to verify content integrity. Using distributed emulation, we demonstrate that this new algorithm has performance comparable to Kademlia if the underlay is unrestricted, and outperforms Kademlia and random walks for various restricted-route topologies. We also show that our algorithm has advantages in terms of availability and fault-tolerance, especially in the presence of malicious participants. Compared to Kademlia, we generally see a larger number of replicas and higher success rates for data retrieval.

Our algorithm has been implemented and released as free software; the release includes the measurement tools and topology generators used for the experiments presented in this paper.

II. Related Work

A DHT imposes structure upon the network underlay by connecting peers to a certain subset of all nodes in the network. The size and method of construction of the routing table is one of the key design choices that distinguish DHTs. For example, Kademlia [11] has routing tables of size $O(\log n)$ and can route requests to the proper destination with $O(\log n)$ steps.

Another key design choice for a DHT is the routing or lookup behavior, which is categorized either as iterative or recursive [6]. In iterative routing, the initiator directly connects to each hop and retrieves information about the next hop until the initiator has a direct connection to the final destination. As a result, the initiator of a request has full control over which node(s) the request is forwarded to at each step — and can possibly tackle problems (such as node failures or malicious participants) during the propagation (for example, by choosing alternative paths).

With recursive routing, the request is forwarded through the network from the first hop onwards according to the routing algorithm and the initiator is only involved again as the final destination of the response, if there is any. A key benefit of recursive routing is that the initiator does not have to be able

to connect to each peer that participates in request routing. However, recursive routing is also less fault-tolerant due to the initiator's lack of control.

A. Kademlia

We use a modified version of Kademlia [11] as the basis of our routing algorithm. The Kademlia algorithm has been shown to work well in networks with common rates of churn [12] and has, in practice, proven capable of handling millions of peers [17]. Kademlia uses XOR to determine the distance between elements in the key space.

Kademlia's routing table is structured as an array of k-buckets. Kademlia uses as many k-buckets as there are bits in the address space. Each k-bucket can hold up to k peers. The i-th k-bucket stores up to k peers whose identifiers are between distance 2^i and 2^{i+1} from the local peer. Routing in Kademlia is iterative; at each step the initiating node picks r closest peers for the next step. Those r peers are queried and return a set of peers closer to the key, and routing continues in this fashion until no closer peers are found. Finally, Kademlia stores data at the r closest peers to the key. Kademlia achieves $O(\log n)$ routing performance: in each step the distance to the destination is at least halved.

One failing of Kademlia is that it has been shown vulnerable to numerous attacks, such as poisoning [10] and Sybil [16] attacks. For example, an adversary may want to deny participants access to a particular key. This can be achieved by creating r peers with identifiers closer than the closest current peer to the key; afterwards, all requests will effectively end at an adversary-controlled peer. Access to the data is then under the control of the adversary.

B. Restricted-Route Topologies

We use the term "restricted-route topology" to refer to a connected underlay topology which prohibits (restricts) direct connections between some of the nodes. Common DHT routing algorithms show diminished performance or even arrant failure when operating over a restricted-route underlay. A common solution on the Internet is to restrict participation in the DHT to peers that are not encumbered by NAT or firewalls. However, this solution limits load-distribution for P2P applications on the Internet and does not work at all for physical networks or friend-to-friend networks. For these types of networks, some other method of routing must be employed to cope with restrictions on direct communication.

C. Freenet

Freenet [15] is the only efficient DHT design we are aware of which works well in restricted-route networks without coordinates. The main problem with Freenet's DHT is the inherent vulnerability of the critical location swapping operation [5]. This operation allows an adversary with only a few peers anywhere in the network to cause massive peer identifier clustering, leading to possible data loss and destroying the load balancing properties of the DHT.

III. Design of R^5N

The basic idea of R^5N is to take advantage of the limited connectivity of restricted-route networks by using the large number of peers that are closer to a key than any of their neighbors for replication. A PUT operation is used to store data at a random subset of these peers, and subsequent GETs then attempt to reach one of the replicas. PUTs are repeated at a certain frequency to refresh data. Since R^5N performs non-deterministic routing, repeated PUTs are likely to result in data being stored at different peers. Furthermore, since our design specifies that this refresh period is significantly shorter than the timeout of content at the replica nodes, this increases the chance of success for subsequent GET operations.

Naturally, a GET may still fail to find its target value. In this case, R^5N expects peers performing GETs to retry a few times. Since routing of GETs is also non-deterministic, repeating the GET operation has a high chance of reaching different peers and hence improves the chance of finding the data. While the GET failure rate is guaranteed to decline over time, the specifics depend on a replication parameter r, the network topology and the number and behavior of adversaries in the network.

Since both GET and PUT operations take different paths each time, an adversary has little chance to successfully place his nodes in the network to block particular key-value pairs. Depending on how the restricted-route underlay is constructed, an isolation attack on nodes may still succeed.

The remainder of this section will detail the various components required for the R^5N routing algorithm. Specifically, we will discuss routing table construction, request processing, content replication and application-level requirements (specifically content validation).

A. The Routing Table

Routing tables in R^5N are constructed and maintained in the same manner as in Kademlia (Section II-A), with the main difference being that R^5N expects that (especially higher numbered) buckets will be empty even though peers with appropriate identifiers exist in the network — direct connections were simply not possible or peers were not discovered because lookups failed (where they would have succeeded in Kademlia). As in Kademlia [11], this results in $O(\log n)$ connections to neighbors. It should be noted that a small difference in routing table maintenance arises indirectly because FIND PEER messages are routed non-deterministically, in the same manner as GET and PUT requests.

B. Routing

Routing in R^5N is recursive and is performed in two distinct phases. In phase one, a request for a key is routed for some number of hops using random neighbors from the routing table. In phase two, routing is deterministic using the peers from the routing table that are closest to the given target. Each request includes the number of hops h that the request has traversed so far, and each peer is supposed to increment the counter by one at each hop. Once the hop counter exceeds a

threshold of $T \approx \log n$ where n is the size of the network, the request enters the second phase. The intuition behind this is that we first make the starting point in the network independent from the location of the initiator and then efficiently find a nearest peer. Assuming the underlay topology is a restricted-route topology, there are many peers that are nearest to the key as far as their immediate neighborhood is concerned:

Lemma 1 (Number of Nearest Peers in a Random Graph)**.** *For a random network with n peers and c random connections per peer, the expected number of nearest peers in the network to any random key is $\frac{n}{c+1}$.*

The optimal number of random hops taken is equal to the mixing time of the graph [8]. The Markov mixing time for various graphs is well known. In a full clique, the optimal number of random steps to take is 1; in a completely random graph, it is $O(\log^2 n)$ steps [1]. For small-world and social networks, the mixing time has been shown empirically to be $O(\log n)$ [2]. Since we expect R^5N to be used primarily in network topologies that more or less conform to small-world topologies, $T \sim \log n$ random hops should be enough to arrive at a sufficiently random point in the graph.

Each request also contains a unique identifier and a 128-bit Bloom filter (each route message is approximately 1k bits in our implementation) which are used to improve efficiency by preventing looping and limiting repeated forwarding of the same request to the same peer. The Bloom filter is updated with the list of peers selected for forwarding the request to at each hop — those peers that match the Bloom filter are excluded from the selection process.

C. Processing Requests and Replies

Each peer that receives a routing request performs the same basic sequence of operations. First, the peer determines whether it is closer to the key of the request than any of the peers in its routing table. If the current peer is a nearest peer, PUT requests are not forwarded; instead the data is stored locally. GET requests where the only possible result is found locally are also not forwarded. Otherwise, the request is forwarded to neighboring peers; these are selected from the routing table using random peer selection or the XOR distance metric depending on the current hop counter. The number of forward replicas is calculated according to the replication level, network size estimate and number of hops traversed so far as described in Section III-D.

For handling replies, each peer tracks a bounded number of active requests, including the respective identity of the preceding peer. Responses are forwarded along the request paths until they reach the original peer or are discarded by a peer that lacks path information (due to memory limitations, for example). It should be noted that most other DHTs do not require this additional state since, in traditional DHTs, the normal routing mechanism can also be used to route replies. For R^5N, this is not feasible due to path randomization. Were R^5N to use randomization for replies, the success rate for replies to reach the intended initiator would be rather low. In contrast, randomization for the lookup is acceptable since many peers are expected to store the data due to replication.

D. Replication

In R^5N, replication is used not only to protect against node failure, but also to improve the chances of a lookup operation finding the desired datum in the absence of failures. For R^5N, the highest GET success rate would be achieved if there are $\frac{n}{c+1}$ replicas in the network (Lemma 1). We use r to describe the desired replication level and for R^5N the target value is $r \sim \sqrt{\frac{n}{c+1}}$; this choice represents a trade-off between the cost for PUTs and the performance for GETs.

If the initiator were to transmit r PUT requests to obtain r replicas, there would be a good chance of collision in the resulting paths and this might be a strong burden on the direct neighbors of the initiator, especially since in the underlay the initiator may not even have r neighbors. Instead, R^5N attempts to have (on average) $1 + \frac{(r-1)h}{T}$ PUT requests active in the network at hop h.

Lemma 2. *Let h be the number of hops in the network that the query has already traversed. If the network is large enough that r random paths of length T are unlikely to merge and if $h < T$, then the average number of peers to which a peer forwards a request to should be*

$$\Upsilon_{r,h} := 1 + \frac{(r-1)}{T + (r-1)h} \tag{1}$$

in order to achieve the desired replication level r at T hops.

A full discussion and proof of this Lemma can be found in [4].

R^5N uses a biased random selection, forwarding to either $\lfloor \Upsilon_{r,h} \rfloor$ or $\lceil \Upsilon_{r,h} \rceil$ peers to reach on average $\Upsilon_{r,h}$ peers for the next hop. We continue to forward to $\Upsilon_{r,h}$ peers for $h \leq 2 \cdot T$ (instead of just until $h < T$) to compensate for path collisions, inaccuracies in the network size prediction and not forwarding PUT requests from nearest neighbors.

E. Content Validation

A key concern for any DHT is the integrity of the content stored in the system. R^5N provides an application with hooks for integrity checks to detect malformed key-value pairs. The application designer writes appropriate content validation functions for discovering malformed key-value pairs. Such pairs are then not forwarded or stored by well-behaved peers, reducing storage and bandwidth requirements in the presence of faulty or malicious participants and making DHT pollution more difficult.

Another possible issue is allowing multiple values to be stored under the same key. Requests in R^5N include a Bloom filter which matches replies already known to the requester. While Bloom filters offer a compact way to filter replies, they can also produce false-positives. R^5N mitigates this problem by having the requester provide an additional 32-bit mutation value which modifies the hash function used for testing the Bloom filter. This alters the bit positions which are set in the

Bloom filter, making it less likely the same previous false positive will match. By re-issuing the request with a different mutation value, these false-positives can be eliminated.

IV. Experimental Results

We've evaluated R^5N for various underlay topologys. In this paper, we will focus on small-world topologies created by extending a 2D-torus by adding or rewiring random links [7]. More extensive experimental results can be found in [4].

Unless stated explicitly otherwise, the presented experiments were done using a fixed replication level of $r = 10$ and a fixed network size estimate parameter $T = 4$, ensuring that only the shape of the topology and the node degree are parameters for the evaluation.

We chose these values for r and T to enable a fair comparison between R-Kademlia and R^5N. Specifically, we chose a value of r that is attainable by R-Kademlia for the various topologies (see Figure 1 for details). Furthermore, as explained in Section III, T is set to correspond roughly to the number of hops required by R-Kademlia.

A. Experimental Setup

We have analyzed the expected performance of R^5N using mathematical analysis, simulation and emulation [3]. Due to space constraints, the experimental results presented in this section are only based on our experiments using emulation. For these experiments, we implemented R^5N atop an existing P2P framework. The results presented in this paper were obtained by emulating 2025 peers on a single desktop, which is close to the limits of our hardware and since $45^2 = 2025$ this number allows for the construction of a clean 2D-torus topology as a starting point for our small-world topology construction. Our emulation does not model network latencies; however, this is not a significant problem since R^5N currently ignores link-latencies in its peer selection strategy.

B. Adversary Model

We consider a number of types of malicious adversaries with diverse goals in our design. We assume that each participating malicious node has similar resources to that of a normal participant in the network, and may eavesdrop, alter, send and receive messages. This enables flooding and poisoning attacks (where specific keys are inserted for purposes of denial-of-service or blocking access). An adversary is assumed to be able to create or impersonate multiple peers running simultaneously with free choice of peer identity for identification and lookup in the DHT. We also assume that adversaries may collude in order to achieve a specific goal, for instance to perform a sybil or eclipse attack. Finally, we assume that encrypted messages intercepted at the network level are unable to be decrypted by peers other than the intended recipient.

While flooding and poisoning attacks can be detrimental to DHTs in general, we do not focus on attacks on data — it entirely depends on the specific application using R^5N and its implementation of the validation hooks (see Section III-E) to address this issue. For this paper, we are primarily concerned with routing-level security.

C. R-Kademlia

We use a variant of Kademlia, which we call R-Kademlia, as a point of comparison with our own algorithm. The iterative routing in the original Kademlia design performs so badly in a restricted-route topology that it is not useful for comparison. R-Kademlia is a recursive implementation of Kademlia that is otherwise as faithful to the original design as possible.

The first — and biggest — problem with a recursive implementation of Kademlia is that r concurrent requests are meant to be kept in flight until no closer peers are found. R-Kademlia initiates r requests at the first peer. These requests terminate once a nearest peer is reached; however, the initiator has no way to guarantee this. Peers in R-Kademlia are responsible for attempting to forward requests only to peers that have not encountered the request already using a Bloom filter (as explained in III-B). Peers also maintain a limited store of recent requests; thus, if the same request reaches a peer twice, the Bloom filters are merged. Using these techniques, we mimic the iterative routing of Kademlia, with the exception that the initiator cannot control the next-hop decisions.

D. Worst-Case Network Performance

For networks with few connections, the success rate of R^5N is significantly higher than it is for R-Kademlia. The worst case for R^5N when compared to R-Kademlia is hence an unrestricted underlay topology (clique). In this case, both R-Kademlia and R^5N will always find the data at the nearest peer on the first attempt, but R^5N is expected to take longer. Table I shows the average number of hops taken for the two designs in this worst-case scenario for R^5N.

TABLE I: Average hops for R-Kademlia and R^5N in clique underlay topologies of different sizes. As expected, R^5N takes about twice as many hops as R-Kademlia.

Size of network	Average hops per PUT		Average hops per GET	
	R-Kademlia	R^5N	R-Kademlia	R^5N
100	2.70 ± 0.06	3.96 ± 0.06	2.54 ± 0.03	4.63 ± 0.17
250	3.06 ± 0.10	4.26 ± 0.10	3.10 ± 0.06	5.96 ± 0.27
500	3.08 ± 0.46	4.38 ± 0.45	3.38 ± 0.06	6.17 ± 1.14
750	3.19 ± 0.74	4.37 ± 0.83	3.50 ± 0.04	6.29 ± 1.04
1000	3.63 ± 0.07	4.47 ± 0.93	3.64 ± 0.04	7.29 ± 0.95

E. Replication Performance

As described in Section IV-C, R-Kademlia attempts to achieve a certain replication level r by starting r requests in parallel from the initiating peer. In contrast, R^5N probabilistically chooses multiple peers to forward the request to at each hop. Neither approach is able to precisely hit the specified replication target; however, R^5N produces the same number of replicas with significantly fewer messages when compared to R-Kademlia (Figure 1) for the small-world topology. This is because sending out many parallel requests from the same initial peer increases the chance that paths will at times converge, while requests that branch at later hops are likely to be further apart in the network and carry more

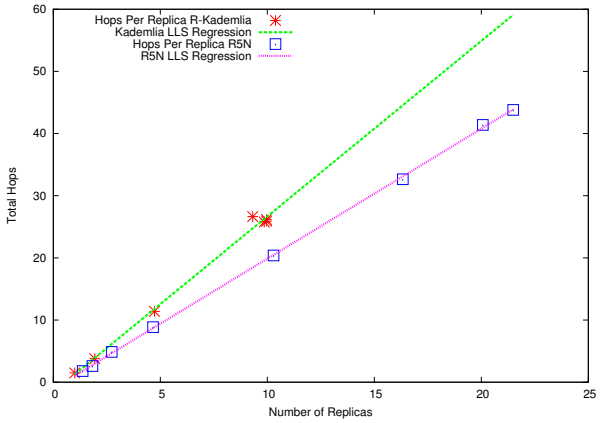

Fig. 1: Average hops required per replica; varying replication level r.

information about which peers have already been routed to and therefore overlap with lower probability. We limit results to 30 total replicas or less; at higher replication levels R^5N outperforms R-Kademlia.[1]

Figure 2 compares the total number of replicas after several rounds of PUT operations for the same key-value pair (without churn or replica expiration). The figure shows the number of replicas that is achieved by either R-Kademlia and R^5N for the case where either the *same* peer performs the PUT operation or where the source of the put operation is chosen at *random* in each round. If the same peer performs the PUT operation using R-Kademlia, the PUT paths always converge at the same nearest peers and, hence, the number of replicas remains constant. In contrast, with R^5N, random peer selection achieves significantly higher levels of replication over time. If PUTs in R-Kademlia are started at a random peer, the resulting replication levels are only slightly higher, suggesting that the random phase achieves its mixing goal.

[1]Due to R-Kademlia's inability to create more replicas than connections.

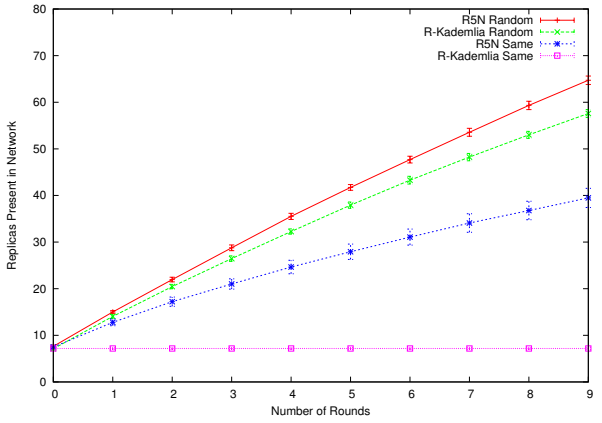

Fig. 2: Replication over time; same starting peer vs. randomized starting peers. Note that standard deviations are quite small due to usage of the same topology in each trial.

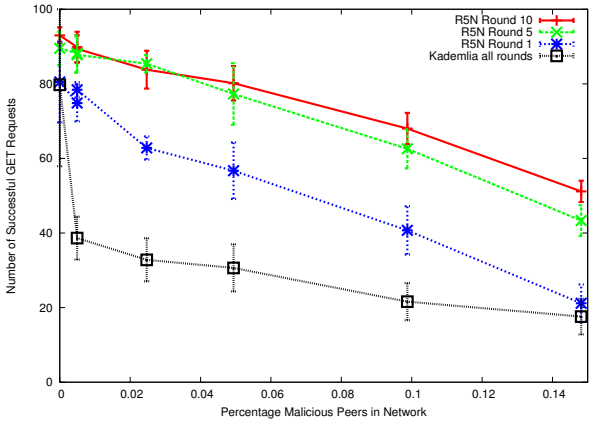

Fig. 3: Number of malicious peers at random locations in a network with 2025 peers vs. percentage of successful GET requests. Each of the μ malicious peers drop all requests they receive, representing the simplest type of malicious participant.

F. Robustness Against Active Adversaries

An additional goal for our routing algorithm is to perform well in the presence of malicious participants. Handling malicious participants well subsumes handling peer failure due to bugs, churn or misconfiguration. The design of R^5N already explicitly addresses malicious peers that attempt to perform denial of service (DoS) attacks on the network by bounding the resource multiplier effect of all operations. Peers cannot send requests that consume significantly more resources than "normal" requests, so an adversary can only multiply its own bandwidth by less than the average number of hops for requests multiplied by the replication level r. Similarly, poisoning attacks may be mitigated using content validation hooks (Section III-E).

An active adversary could also join the network with peers that simply passively drop all requests that are received. For these experiments, we vary μ, the number of malicious adversaries which drop requests. This kind of attack is already quite detrimental to overall operation for deterministic algorithms: any request that traverses any of the malicious peers fails. R-Kademlia's redundancy (r-replication) is a typical mitigation strategy. Figure 3 shows the impact of a dropping adversary on the performance of R^5N and R-Kademlia in terms of success rates for GET operations (initiated at peers selected uniformly at random in each round) for a small-world topology generated to have 2025 nodes and 30k edges. The GET operations were performed after a number of rounds of PUT operations which are initiated at the same peer in each round.

Later GET rounds in R^5N have higher success rates because additional PUT rounds increase availability for R^5N as more replicas are created. The benefit of R^5N over R-Kademlia is clearly seen in the small-world topology where R^5N achieves significantly better performance under this type of attacker. Results for other topologies can be found in [4].

We now consider an attacker trying to prevent access to a particular key using an Eclipse attack. The attacker again

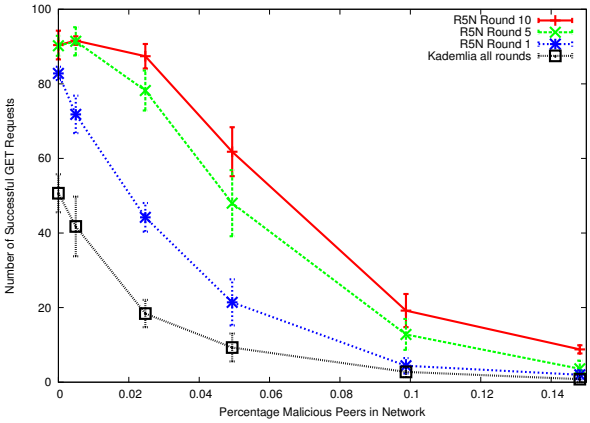

Fig. 4: Number of malicious peers present at sybil locations in a network with 2025 peers vs. percentage of successful GET requests.

simply drops all GET and PUT requests; however, this time the μ malicious nodes are not placed into the network at random but at the μ peers that are closest to the key. This represents an attacker performing a Sybil attack with free choice of identifier and node placement in a restricted-route topology — the strongest type of Sybil attacker we can imagine. This attack has a serious impact on Kademlia-based DHTs [16].

While additional traditional protections against Sybil attacks could be deployed to further mitigate this attack [4], Figure 4 shows that such measures may be unnecessary for R^5N. Again, as rounds of PUT requests increase the number of replicas in the network, R^5N's success rate increases. Again, R^5N outperforms R-Kademlia and is especially strong in the case of a Sybil attack on the small-world underlay topology, where even the first round of GET requests succeeds with a much higher rate than R-Kademlia.

V. Performance Analysis

To achieve high success rates, R^5N needs to create a sufficient number of replicas. A network with n nodes of degree c is expected to have $\frac{n}{c-1}$ nearest peers. Assuming that T is chosen large enough to achieve perfect mixing, $\sqrt{\frac{n}{c-1}}$ replicas would need to be created in order for a GET request to succeed with about 50% probability according to the birthday paradox. Once individual requests succeed with this probability, a small constant number of repetitions can be used to get high overall success rates. As we have shown experimentally, the relationship between the number of replicas in the network and the number of hops required for the respective PUT operations is almost linear (Figure 2). Since individual PUT and GET requests have complexity $O(\log n)$, routing in small-world networks using R^5N scales with $O(\sqrt{n} \cdot \log n)$. Note that this does not hold in sparse graphs with large diameter or graphs that are not expander graphs (such as a circle) because the routing tables could not be sufficiently populated.

VI. Conclusion

We have presented a robust routing algorithm for restricted-route networks. Our R^5N algorithm combines a random walk with a recursive variation of Kademlia and uses forwarding to multiple targets along the path for replication and redundancy. R^5N has good performance and is robust against a range of some well-known attacks on DHTs, including poisoning attacks, Sybil attacks and Eclipse attacks.

Acknowledgments

This work was funded by nlnet and the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1.

References

[1] C. Avin and G. Ercal, "On the cover time and mixing time of random geometric graphs," *Theor. Comput. Sci*, 2007.

[2] M. Dell amico and Y. Roudier, "A measurement of mixing time in social networks," in *5th International Workshop on Security and Trust Management*, Saint Malo, France, September 2009.

[3] N. Evans and C. Grothoff, "Beyond simulation: Large-scale distributed emulation of p2p protocols," in *4th Workshop on Cyber Security Experimentation and Test (CSET 2011)*. USENIX Association, 2011.

[4] N. S. Evans, "Methods for secure decentralized routing in open networks," Ph.D. dissertation, Technische Universität München, 2011.

[5] N. S. Evans, C. GauthierDickey, and C. Grothoff, "Routing in the dark: Pitch black," in *23rd Annual Computer Security Applications Conference*. IEEE Computer Society, 2007, pp. 305–314.

[6] J. Hautakorpi and G. Camarillo, "Evaluation of dhts from the viewpoint of interpersonal communications," in *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*. ACM, 2007, pp. 74–83.

[7] J. M. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, pp. 845–845, 2000.

[8] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov chains and mixing times*. American Mathematical Society, 2006.

[9] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. F. Kaashoek, "Comparing the performance of distributed hash tables under churn," in *Proc. of the 3rd IPTPS*, 2004.

[10] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer, "Poisoning the Kad Network," in *11th International Conference on Distributed Computing and Networking (ICDCN), Kolkata, India*. Springer, January 2010, pp. 195–206.

[11] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *1st International Workshop on Peer-to Peer Systems*, Cambridge, March 2002, pp. 53–65.

[12] Z. Ou, E. Harjula, O. Kassinen, and M. Ylianttila, "Performance evaluation of a kademlia-based communication-oriented p2p system under churn," *Comput. Netw.*, vol. 54, pp. 689–705, April 2010.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, August 2001.

[14] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[15] O. Sandberg, "Distributed routing in small-world networks," in *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments*, 2006, pp. 144–155.

[16] M. Steiner, T. En-najjary, and E. W. Biersack, "Exploiting kad: possible uses and misuses." *Computer Communication Review*, vol. 37, no. 5, pp. 65–70, October 2007.

[17] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of kad," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 117–122.

[18] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM on Internet measurement*. New York, NY, USA: ACM Press, 2006, pp. 189–202.

[19] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: Defending against sybil attacks via social networks," in *The ACM SIGCOMM'06 Conference*. ACM Press, 2006, pp. 267–278.