Routing in the Dark: Pitch Black

Anonymous

Abstract. In many networks, such as mobile ad-hoc networks and friend-to-friend overlay networks (darknets), direct communication between nodes is limited to specific neighbors. Often these networks have a small-world topology; while short paths exist between any pair of nodes in small-world networks, it is non-trivial to determine such paths with a distributed algorithm. Recently, Clarke and Sandberg proposed the first decentralized routing algorithm that achieves efficient routing in such small-world networks.

This paper is the first independent security analysis of Clarke and Sandberg's routing algorithm. We show that a relatively weak participating adversary can render the overlay ineffective without being detected, resulting in significant data loss due to the resulting load imbalance. We have measured the impact of the attack in a testbed of 800 nodes using minor modifications to Clarke and Sandberg's implementation of their routing algorithm in Freenet. Our experiments show that the attack is highly effective, allowing a small number of malicious nodes to cause rapid loss of data on the entire network.

We also discuss various proposed countermeasures designed to detect, thwart or limit the attack. While we were unable to find effective countermeasures, we hope that the presented analysis will be a first step towards the design of secure distributed routing algorithms for restricted-route topologies.

1 Introduction

Fully decentralized and efficient routing algorithms for restricted route networks promise to solve crucial problems for a wide variety of networking applications. Efficient decentralized routing is important for sensor and general wireless networks, peer-to-peer overlay networks and theoretically even next generation Internet (IP) routing. A number of distributed peer-to-peer routing protocols developed in recent years achieve scalable and efficient routing by constructing a structured overlay topology [4,5,8,11,15]. However, all of these designs are unable to work in real-world networks with *restricted routes*. In a restricted route topology, nodes can only directly communicate with a subset of other nodes in the network. Such restrictions arise from a variety of sources, such as physical limitations of the communications infrastructure (wireless signals, physical network topology), policies (firewalls) or limitations of underlying protocols (NAT, IPv6-IPv4 interaction).

Recently, a new routing algorithm for restricted route topologies was proposed [13] and implemented in version 0.7 of Freenet, an anonymous peer-to-peer file-sharing network [3]. The proposed algorithm achieves routing in expected

$O(\log n)$ hops for small-world networks with n nodes by having nodes swap *locations* in the overlay under certain conditions. This significant achievement raises the question of whether the algorithm is *robust* enough to become the foundation for the large domain of routing in restricted route networks.

The research presented in this paper shows that any participating node can severely degrade the performance of the routing algorithm by changing the way it participates in the location swapping aspect of the protocol. Most of the guards in the existing routing implementation are ineffective or severely limited and in particular fail to reliably detect the malicious nodes. Experiments using a Freenet testbed show that a small fraction of malicious nodes can dramatically degenerate routing performance and cause massive content loss in a short period of time. Our research also illuminates why churn impacts the structure of the overlay negatively, a phenomenon that was observed by Freenet users in practice but has, to the best of our knowledge, never been explained.

The paper is structured as follows. Section 2 describes related work focusing on distributed hash tables for networks without restricted-route topologies and gives details about small-world networks, a particularly common form of a restricted-route topology. Section 3 details Freenet's distributed darknet routing algorithm for small-world networks. The proposed attack is described in Section 4, followed by experimental results showing the effects of the attack in Section 5. Possible defenses and their limitations are discussed in Section 6.

2 Related Work

2.1 Distributed hash tables

A distributed hash table is a data structure that enables efficient key-based lookup of data in a peer-to-peer overlay network. Generally, the participating peers maintain connections to a relatively small subset of the other participants in the overlay. Each peer is responsible for storing a subset of the key-value pairs and for routing requests to other peers. In other words, a key property of DHTs in a peer-to-peer setting is the need to route queries in a network over multiple hops based on limited knowledge about which peers exist in the overlay network. Part of the DHT protocol definition is thus concerned with maintaining the structure of the network as peers join or leave the overlay.

DHT designs can be characterized using the performance metrics given in Table 1. Routing in DHTs is generally done in a greedy fashion and resembles lookups in skip lists [10]. Table 2 summarizes the key properties of various existing DHT designs. The table does not capture properties which are hard to quantify, such as fault-tolerance. Most existing DHT designs achieve near perfect load balancing between peers. Hosts that can provide significantly more resources than others are usually accommodated by associating multiple locations in the overlay with a single host. In some sense, those hosts are counted as multiple peers.

A major limitation of the DHT designs listed in Table 2 is that they do not support routing in restricted route topologies. These DHTs assume that it

(1)	Messages required for each key lookup
(2)	Messages required for each store operation
(3)	Messages needed to integrate a new peer
(4)	Messages needed to manage a peer leaving
(5)	Number of connections maintained per peer
(6)	Topology can be adjusted to minimize per-hop latency (yes/no)
(7)	Connections are symmetric or asymmetric

Table 1. Performance metrics for DHTs.

	Chord [15]	Pastry [12]	Kademlia [8]	CAN [11]	RSG [5]
(1)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n^{-d})$	$O(\log n)$
(2)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n^{-d})$	$O(\log n)$
(3)	$O(\log^2 n)$	$O(\log n)$	$O(\log n)$	$O(d + n^{-d})$	$O(\log n)$
(4)	$O(\log^2 n)$	$O(1)$	$O(1)$	$O(d)$	$O(\log n)$
(5)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(d)$	$O(1)$
(6)	no	yes	yes	yes	no
(7)	asymmetric	asymmetric	symmetric	symmetric	asymmetric

Table 2. Comparison of DHT designs. The numbers refer to the list of performance metrics given in Table 1. The value d is a system parameter for CAN.

is generally possible for any peer to connect to any other peer. However, firewalls and network address translation (NAT) make this assumption unrealistic over the current Internet where large-scale studies have shown that over 70% of machines are NATed [1].

In contrast to the DHT designs from Table 2, the Freenet routing algorithm achieves expected $O(\log n)$ routing in restricted route topologies under the assumption that the restricted network topology has small-world properties.

2.2 Small-world networks

A small-world network is informally defined as a network where the average shortest path between any two nodes is "small" compared to the size of the network, where "small" is generally considered to mean at least *logarithmic* in relation to the size of the network. Small world networks occur frequently in the real world [16], the most prominent example being social networks [9].

Watts and Strogatz [16] characterized small-world networks as an intermediate stage between completely structured networks and random networks. According to their definition, small world networks with n nodes have on average k edges per vertex where $n >> k >> \log n$. They define a *clustering coefficient* which captures the amount of structure (clustering) in a given network. Small-world networks are then networks with a clustering coefficient significantly larger than the coefficients of completely random networks and with average shortest

path lengths close to those for of completely random networks. Watts and Strogatz's work explains why short paths exist in real-world networks.

Kleinberg [6,7] generalized Watts and Strogatz' construction of small-world networks and gave sufficient and necessary conditions for the existence of efficient distributed routing algorithms for these constructions. Kleinberg's model for distributed routing algorithms does not include the possibility of nodes swapping locations, which is a fundamental part of Freenet's "Darknet" routing algorithm.

3 Freenet's "Darknet" routing algorithm

Freenet [3] is a peer-to-peer network where the operator of each node specifies which other peers are allowed to connect to the node [2]. The main reason for this is to obscure the participation of a node in the network – each node is only directly visible to the friends of its' operator. Peer-to-peer networks that limit connections to friend-to-friend interactions are often called *darknets*. Given that social networks are small-world networks and that small-world networks arise easily given a certain amount of "randomness" in the graph construction, it is realistic to assume that Freenet's darknet is a small-world network. The routing restrictions imposed on the Freenet overlay could technically model arbitrary network limitations; consequently, an efficient distributed routing algorithm for such a topology should easily generalize to any small-world network.

3.1 Network creation

The graph of the Freenet network consists of vertices, which are peers, and edges, which are created by friend relationships. An edge *only* exists between peers if both operators have agreed to the connection a priori. Freenet assumes that a sufficient number of edges (or friend relationships) between peers will exist so that the network will be connected.

Each Freenet node is created with a unique, immutable *identifier* and a randomly generated initial *location*. The identifier is used by operators to specify which connections are allowed while the location is used for routing. The location space has a range of $[0, 1)$ and is cyclic with 0 and 1 being the same point. For example, the distance between nodes at locations 0.1 and 0.9 is 0.2.

Data stored in the Freenet network is associated with a specific *key* from the range of the location space. The routing algorithm transmits `get` and `put` requests from node A to the neighbors of A starting with the neighbor with the closest location to the key.

3.2 Operational overview

The basic strategy of the routing algorithm is to greedily forward a request to the neighbor whose location is closest to the key. However, the simple greedy forwarding is not guaranteed to find the closest peer – initially, the location of each peer is completely random and connections between peers are restricted

(since a peer is only allowed to establish connections the operator explicitly designated as allowed). Consequently, the basic greedy algorithm is extended to a depth-first search of the topology (with bounded depth) where the order of the traversal is determined by the distance of the nodes to the key [14]. Figure 1 shows the routing algorithm for the `get` operation in pseudocode.

For `put` operations, Freenet creates a copy of the content in the local datastore of every node on the path from the initiator to the peer with the closest location with respect to the key as found by greedy routing. The `put` algorithm is detailed in Figure 2.

1. Check that new `get` request is not identical with recently processed request; if the request is a duplicate, notify sender about duplication status, otherwise continue.
2. Check local data store for data; if data is found, send response to sender, otherwise continue.
3. If hops-to-live of request is zero, respond with data not found, otherwise continue.
4. Find closest neighbor (in terms of peer location) with respect to the key of the `get` request, excluding those routed to already. Forward the request to the closest peer with (probabilistically) decremented hops-to-live counter. If valid content is found, forward content to sender, otherwise, repeat step 4.

Fig. 1. Pseudocode for routing of a `get` request.

1. Check that new `put` request is not identical to a recently processed request; if the request is a duplicate, notify sender about duplication status, otherwise continue.
2. Insert the data into the local datastore.
3. Find closest neighbor location to key, if the closest location is the location of the current node, then reset hops-to-live to the maximum hops-to-live and forward the `put` request to all neighbors.
4. If hops-to-live is zero, signal success to sender, otherwise forward the request to the nearest peer.

Fig. 2. Pseudocode for routing of a `put` request.

3.3 Location swapping

To make the routing algorithm find the data faster, Freenet attempts to cluster nodes with similar locations. The network achieves this by having nodes swap their locations under certain conditions:

1. Let $L(n)$ be the current location of node n.

2. A node A randomly chooses a neighboring node B and initiates a swap request. Both nodes exchange the locations of their respective neighbors and calculate $D_1(A, B)$, the product of the products of the differences of the locations of them to their neighbors':

$$D_1(A,B) = \prod_{(A,n)\in E} |L(A) - L(n)| \cdot \prod_{(B,n)\in E} |L(B) - L(n)| \tag{1}$$

3. The nodes also compute $D_2(A, B)$, the product of the products of the differences of the locations of them to their neighbors' *after* a potential swap:

$$D_2(A,B) = \prod_{(A,n)\in E} |L(B) - L(n)| \cdot \prod_{(B,n)\in E} |L(A) - L(n)| \tag{2}$$

4. If the nodes find that $D_2(A, B) \leq D_1(A, B)$, they swap locations, otherwise they swap locations with probability $\frac{D_1(A,B)}{D_2(A,B)}$.[1]

As a result of this swapping of locations the overlay becomes semi-structured; the routing algorithm's depth first search can utilize this structure in order to find short paths with high probability. Sandberg's thesis [13] shows that the Freenet routing algorithm converges towards routing in $O(\log n)$ steps (with high probability) under the assumption that the set of legal connections specified by the node operators forms a small-world network. This is a significant result because it describes the first fully decentralized distributed hash table (DHT) design that achieves $O(\log n)$ routing with (severely) restricted routes. Most other DHT designs make the unrealistic assumption that every node is able to directly communicate with every other node [5,8,12,15].

3.4 Content Storage

Each Freenet node stores content in a datastore of bounded size. Freenet uses a least-recently-used content replacement policy, removing the least-recently-used content when necessary to keep the size of the datastore below the user-specified limit.

Our simulation of routing in the Freenet network places the content at the node whose location is closest to the key and does not allow caching or replication of content. The reason for this is that our study focuses on routing performance and not on content replication and caching strategies.

3.5 Example

Figure 3 shows a small example network. Each node is labeled with its location ($L_n \in [0, 1)$) in the network. The bi-directional edges indicate the direct connections between nodes. In a friend-to-friend network, these are the connections

[1] Note that the Freenet implementation fails to use a secure multiparty computation for this probabilistic step and that the nodes perform the swap based on either node requesting to swap. This is an implementation limitation and not a fundamental issue with the proposed routing algorithm.

that were specifically allowed by the individual node operators. Furthermore, each node is only aware of its immediate neighbors. Similarly, in an ad-hoc wireless network, the edges would indicate which nodes are physically able to directly communicate with which other nodes. While our example network lacks cycles, any connected graph is allowed; the small-world property is only required to achieve $O(\log n)$ routing performance, the algorithm itself works for any connected graph.

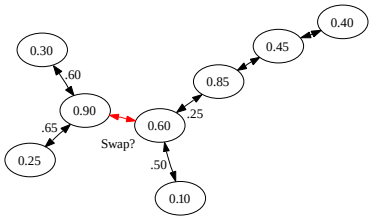

Fig. 3. This figure shows an example network with two nodes considering a swap. The result of the swap equation is D_1 = .60 * .65 * .25 * .50 = .04875 and D_2 = .30 * .35 * .05 * .80 = .0042. Since $D_1 > D_2$, they swap.

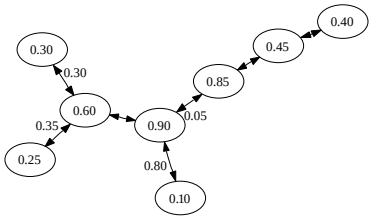

Fig. 4. This figure shows the resulting example network after the swap has occurred.

The network illustrated in Figure 3 happens to have an assignment of locations that would cause the nodes with locations 0.60 and 0.90 to perform a swap in order to minimize the distance product from Equation (1). Figure 4 shows the new assignment of locations after the swap. Note that after a swap each node retains exactly the same set of connections; the only change is in the location identifiers of the nodes. This change in node locations impacts the order of the traversal during routing.

Figure 5 shows how a GET request would be routed after the swap. Starting at the node with location 0.90 and targeting the key 0.23 the node picks its closest neighbor (with respect to the key), which is 0.10. However, 0.10 does not have the content and also lacks other neighbors to try and thus responds with

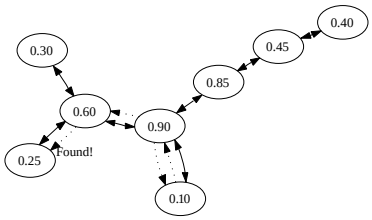

Fig. 5. Illustrates the path of a GET request looking for data with a hash value of .23 which is stored at the node identified by the location 0.25. The GET request is initiated from node with location 0.90. The path that the GET request travels is displayed as the dotted lines which travel from 0.90 → 0.10 → 0.90 → 0.60 → 0.25 where the data is found.

content not found. Then 0.90 attempts its' second-closest neighbor, 0.60. Again, 0.60 does not have the content, but it has other neighbors. The 0.25 neighbor is closest to 0.23. The content is found at that node and returned via 0.60 (the restricted-route topology does not allow 0.25 to send the result directly back to 0.90).

Finally, Figure 6 illustrates how Freenet routes a PUT request. The algorithm again attempts to find the node with the closest location in a greedy fashion. Once a node C is found where all neighbors are further away from the node, the algorithm essentially re-starts the PUT with all neighbors of that node. Usually, these neighbors of C do not have other neighbors that would be closer to the key and thus directly route back to C, which ends the process since C has seen the request already.

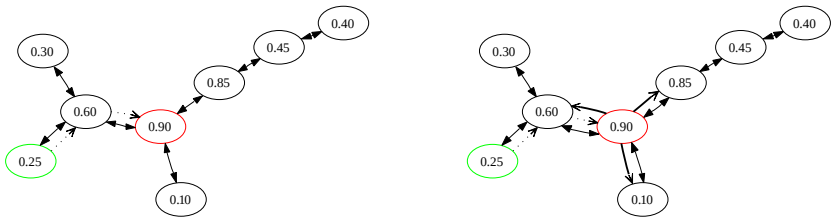

Fig. 6. The graph on the left illustrates the path of a PUT request inserting data with a hash value of .96. The request is initiated from node with location 0.25. The path that the PUT request travels is displayed as the dotted lines which travel from 0.25 → 0.60 → 0.90 where the data is stored. The graph on the right shows what happens after a PUT has found a node whose neighbors are all further away from the key. That node resets the hops-to-live of the PUT request to its maximum and then forwards the PUT request to all of its neighbors. Usually, as in this case, these neighbors have no node closer to the key than their predecessor and the PUT routing ends.

4 Security Analysis

The routing algorithm works under the assumption that the distribution of the keys and peer locations is random. In that case, the load is balanced, in particular all nodes are expected to store roughly the same amount of content and all nodes are expected to receive roughly equivalent numbers of requests.

The basic idea behind the attack is to de-randomize the distribution of the node locations. The attacker tries to cluster the locations around a particular small set of values. Since the distribution of the keys is still random and independent of the distribution of the node locations, clustering of node locations around particular values results in an uneven load distribution. Nodes within the clusters are responsible for less content (because many other nodes are also close to the same set of keys), whereas the load for nodes outside of the clusters is disproportionately high.

We will now detail two scenarios which destroy the initial random distribution of node locations resulting in clustering of locations around particular values. The first scenario uses attack nodes inside of the network. This attack quickly unbalances the load in the network, causing significant data loss; the reason for the data loss is that the imbalance causes some nodes to greatly exceed their storage capacity whereas other nodes store nothing. The other scenario illustrates how location imbalance can occur naturally even without an adversary due to churn.

4.1 Active Attack

As described in Section 3.3, a Freenet node attempts to swap with random peers periodically. Suppose that an attacker wants to bias the location distribution towards a particular location. In order to facilitate the attack, the attacker assumes that particular location. This malicious behavior cannot be detected by the nodes neighbors because the attacker can claim to have obtained this location from swapping. A neighbor cannot verify whether or not such a swap has occurred because the friend-to-friend (F2F) topology restricts communication to immediate neighbors.

The node then creates swap requests in accordance with the Freenet protocol. The only difference is that when the neighbor involved in the swap asks for the locations of the attacker's other neighbors, the attacker responds with locations that are favorable towards swapping. Again, the F2F topology prevents the neighbor involved in the swap from checking the validity of this information. After the swap, the attack node again assumes the original location chosen and continues to try to swap with its other neighbors whose locations are still random.

The neighbors that have swapped with an attacker then continue to swap in accordance with the swapping algorithm, possibly spreading the malicious locations. Once the location has been spread, the adversary subjects another neighbor to a swap, removing yet another random location from the network. The likelihood of neighbors spreading the malicious location by swapping can be

improved by using multiple attack locations. Using a higher number of malicious locations increases the rate of the penetration of the network. There is a tradeoff between the speed of penetration and the impact of the attack in terms of causing load imbalances.

4.2 Natural Churn

Network churn, the joining and leaving of nodes in the network, is a crucial issue that any peer-to-peer routing protocol needs to address. We have ignored churn until now because the attack described in the previous section does not require it. Intuition may suggest that natural churn may help the network against the attack by supplying a constant influx of fresh, truly random locations. This section illustrates that the opposite is the case: natural churn can strengthen the attack and even degenerate the Freenet network in the same manner without the presence of malicious nodes.

For the purpose of this discussion, we need to distinguish two types of churn. The first kind, *leave-join churn*, describes fluctuations in peer availability due to a peer leaving the nextwork *for a while* and then joining again. In this case, the network has to cope with *temporary* loss of availability in terms connectivity and access to content stored at the node. Freenet's use of content replication and its routing algorithm are well-suited to handle this type of churn. Most importantly, a node leaving does not immediately trigger significant changes at any other node. As a result, an adversary cannot use leave-join churn to disrupt network operations. Since honest Freenet peers re-join with the same location that they last had when they left the network, leave-join churn does not impact the overall distribution of locations in the network.

The second kind, *join-leave churn*, describes peers who join the network and then leave *for good*. In this case, the network has to cope with the *permanent* loss of data stored at this peer. In the absence of adversaries, join-leave churn may be less common in peer-to-peer networks; however, it is always possible for users to discontinue using a particular application. Also, often users may just test an application once and decide that it does not meet their needs. Again, we believe that Freenet's content replication is likely sufficient to avoid significant loss of content due to realistic amounts of join-leave churn.

However, natural join-leave churn has another, rather unexpected impact on the distribution of locations in the Freenet overlay. This additional impact requires that the overlay has a stable core of peers that are highly available and strongly interconnected, which is a common phenomeonon in most peer-to-peer networks. In contrast to this set of stable core-peers, peers that contribute to join-leave churn are likely to participate only briefly and have relatively few connections. Suppose the locations γ_i of the core-peers are initially biased towards (or clustered around) a location $\alpha \in [0, 1)$. Furthermore, suppose that (over time) thousands of peers with few connections (located at the fringe of the network) contribute to join-leave churn.

Each of these fringe-peers will initially assign itself a random location $\beta \in [0, 1)$. In some cases, this random choice β will be closer to α than some of the

γ_i-locations of the core nodes. In that case, the routing algorithm is likely to swap locations between these fringe-peers and core-peers in order to reduce the overall distances to neighbors (as calculated according to Equation (1)). Peers in the core have neighbors close to α, so exchanging one of the γ_i's for β will reduce their overall distances. The fringe peers are likely to have few connections to the core group and thus the overall product after a swap is likely to decrease.

Consequently, non-adversarial join-leave churn strengthens any existing bias in the distribution of locations among the long-lived peers. The long-term results of join-leave churn are equivalent to what the attack from Section 4.1 is able to produce quickly – most peers end up with locations clustering around a few values. Note that this phenomenon has been observed by Freenet users and was reported to the Freenet developers – who so far have failed to explain the cause of this degeneration.[2] Since both the attack and natural churn have essentially the same implications for the routing algorithm, the performance implications established by our experimental results (Section 5) hold for both scenarios.

5 Experimental Results

This section presents experimental results obtained from a Freenet testbed with up to 800 active nodes. The testbed, consisting of up to 18 GNU/Linux machines, runs the actual Freenet 0.7 code. The nodes are connected to form a small-world topology (using Kleinberg's 2d-grid model [6]) with on average $O(\log^2 n)$ connections per node.

Each experiment consists of a number of iterations. In each iteration, nodes are given a fixed amount of time to swap locations. Then the performance of the network is evaluated. The main performance metrics are the average path length needed to find the node that is responsible for a particular key, and the percentage of the content originally available in the network that can still be retrieved.

All nodes are configured with the same amount of storage space. Before each experiment, the network is seeded with content with a random key distribution. The amount of content is fixed at a quarter of the storage capacity of the entire network. The content is always (initially and after each iteration) placed at the node with the closest location to the key. Nodes discard content if they do not have sufficient space. Discarded content is lost for the duration of the experiment.

Depending on the goals of the experiment, certain nodes are switched into attack mode starting at a particular iteration. The attacking nodes are randomly chosen, and behave exactly as all of the other nodes, except for aggressively propagating malicious node locations when swapping.

[2] `https://bugs.freenetproject.org/view.php?id=647`, April 2007. We suspect that the clustering around 0.0 is caused by software bugs, resulting in an initial bias for this particular value, which is then strengthened by churn.

5.1 Distribution of Node Locations

Figure 7 illustrates the distribution of node locations on a circle before and after an attack. The initial distribution on the left side consists of 800 randomly chosen locations, which are largely evenly distributed over the entire interval.

The distribution on the right side shows the distribution after eight nodes began an attack on the network in an attempt to create eight clusters. Note that the number of attackers and the number of cluster locations can be chosen independently.

Both plots use thicker dots in order to highlight spots where many peers are in close proximity. Particularly after the attack peers often have locations that are so close to each other (at the order of 2^{-30}) that a simple plot of the individual locations would just show a single dot. Thicker dots illustrate the number of peers in close proximity, the spread of their locations is actually much smaller than the thickness may suggest.

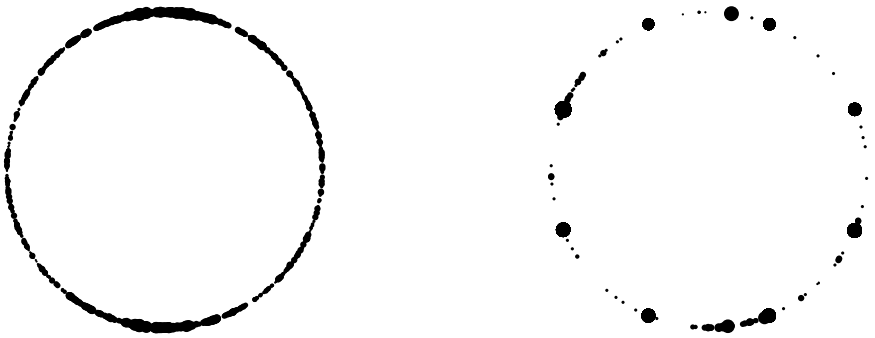

Fig. 7. Plot of node locations before and after attack.

5.2 Availability of Content

Figures 8, 9 and 10 show the data loss in a simulated Freenet network with 800 nodes and two, four and eight attackers respectively. The attackers attempt to use swapping in order to cluster the locations of nodes in the network around eight pre-determined values. The resulting clustering of many nodes around particular locations causes the remaining nodes to be responsible for disproportionately large areas in the key space. If this content assignment requires a particular node to store more content than the node has space available for, content is lost.

The attack is initiated after 75 iterations of ordinary network operation. After just 200 rounds (corresponding to a roughly 5 hours of actual execution time) the network has lost on average between 15% and 60% of its content, depending on the number of attackers. Note that in our model, an individual attacker is granted precisely the same resources as any ordinary user. The figures show the average data loss (and standard deviations) over five runs. For each run, the positions of the attackers were chosen randomly among the 800 nodes.

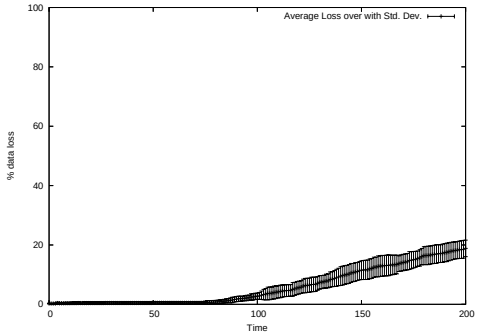

Fig. 8. Graph showing average data loss over 5 runs with 800 total nodes and 2 attack nodes using 8 bad locations with the attack starting after about 2h.

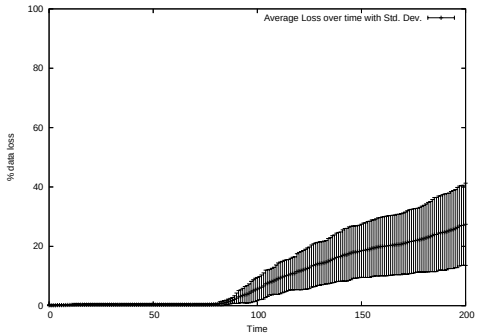

Fig. 9. Graph showing average data loss over 5 runs with 800 total nodes and 4 attack nodes using 8 bad locations with the attack starting after about 2h.

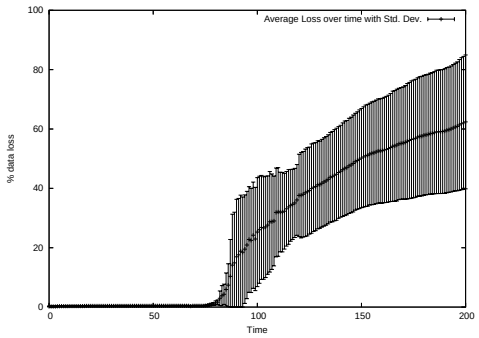

Fig. 10. Graph showing average data loss over 5 runs with 800 total nodes and 8 attack nodes using 8 bad locations with the attack starting after about 2h.

6 Discussion

Various strategies could be used to limit the impact of the proposed attack, including changing the swapping policy, malicious node detection, and secure multiparty communication. While some of these strategies can reduce the impact of the attack, we do not believe that adopting any of the suggested measures would address the attack in a satisfactory manner.

One possibility for reducing the effect of the attack proposed in this paper is to increase the amount of time between attempts to swap, or to have each node in the network periodically reset its location to a random value. The idea is that the malicious node locations would spread more slowly and be eventually discarded. However, while this limits the impact of the attack, this defense also slows and limits the progress of the network converging to the most fortuitous topology.

However, the negative impact of churn may be handled by swapping locations only with long lived peers. Recent measurement studies in peer-to-peer networks have shown a power-law distribution of the uptime of peers; a large percentage of peers have a short uptime. By adjusting the probability of location swapping to be proportional to a peer's uptime, the network may be able to prevent clustering of the locations of long-lived peers due to join-leave churn.

Another possible method attempts to detect a malicious node based on knowing the size of the network. If a Freenet node were able to accurately produce a close estimation of the size of the network, it could detect if an attacker was swapping locations that are significantly closer than what would be likely with a random distribution of locations. The problem with this approach is that in an open F2F network it is difficult to reliably estimate the network's size.

If there were a way for a node which purported to have a certain number of friends to prove that all those friends existed, nodes could be more confident about swapping. The Freenet developers suggested using a secure multiparty computation as a way for a node to prove that it has n connections. The idea would be for the swapping peers to exchange the the results of a computation that could only be performed by their respective neighbors. But because nodes can only directly communicate with their peers (F2F), any such computation could easily be faked given appropriate computational resources. Of course, if a node could directly communicate with another node's neighbors, then the topology could be discerned. However, in that case the protocol no longer works for restricted-route networks.

7 Conclusion

The new Freenet routing algorithm is unable to provide performance guarantees in networks where adversaries are able to participate. The algorithm also degenerates over time (even without active adversaries) if the network experiences churn. The recommended approach to address both problems is to periodically reset the locations of peers. While this limits the deterioration of the routes

through adversaries and churn, such resets also sacrifice the potential convergence towards highly efficient routes. Secure and efficient routing in restricted route networks remains an open problem.

Acknowledgements
The authors thank Anonymous for feedback on an earlier draft of the paper.

References

1. Martin Casado and Michael J. Freedman. Illuminating the shadows: Opportunistic network and web measurment. http://illuminati.coralcdn.org/stats/, December 2006.
2. Ian Clarke. The freenet project. http://freenetproject.org/, 2007.
3. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*. International Computer Science Institute, 2000.
4. Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.
5. Michael T. Goodrich, Michael J. Nelson, and Jonathan Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393, New York, NY, USA, 2006. ACM Press.
6. Jon M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, August 2000.
7. Jon M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, New York, NY, USA, 2000. ACM Press.
8. Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02, Cambridge*, March 2002.
9. Stanley Milgram. The small-world problem. *Psychology Today*, pages 60–67, 1967.
10. William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.
11. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, Berkeley, CA, 2000.
12. Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218, 2001.
13. Oskar Sandberg. *Searching in a Small World*. PhD thesis, University of Gothenburg and Chalmers Technical University, 2005.
14. Oskar Sandberg. Distributed routing in small-world networks. In *ALENEX*, 2006.
15. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
16. Duncan Watts and Steve Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.