Reroute on Loop in Anonymous Peer-to-Peer Content Sharing Networks

Guanyu Tian
Fontbonne University
gtian@fontbonne.edu

Zhenhai Duan
Florida State University
duan@cs.fsu.edu

Todd Baumeister, Yingfei Dong
University of Hawaii
{*baumeist, yingfei*}*@hawaii.edu*

Abstract—**Detecting and handling routing loops is a critical issue in the design of anonymous peer-to-peer content sharing networks (apCSNs). A principal requirement of such a scheme is that it should not leak any message forwarding information that can undermine the user anonymity of the resulting apCSN. A number of loop handling schemes have been developed in apCSNs such as Freenet and GNUnet. However, they all leak certain level of message forwarding information, which can be exploited to undermine or compromise the user anonymity. In this paper we develop a new loop handling scheme named Reroute-on-Loop (ROL) that will not leak any message forwarding information. Using the Thynix simulator developed by the Freenet project, we show that overall ROL only has minor performance impacts on message path lengths compared to the current loop handling scheme of Freenet on various network topologies, ranging from small-world topologies to random topologies. For example, the average routing path lengths of ROL are only increased by less than 1 hop compared to those with the current loop handling scheme of Freenet on small-world network topologies. Our performance studies confirm that ROL is a practical scheme, and can be deployed on Freenet and similar apCSN systems.**

I. Introduction

In order to support censorship-resistant content publishing and user privacy on the Internet, a number of anonymous peer-to-peer content sharing networks (apCSNs) have been developed and deployed, including Freenet and GNUnet [1], [2], [3], [4]. One of the critical problems in the design of such a system is how to detect and handle routing loops. At the high level, two different approaches have been developed. The first one targets proactive routing loop prevention. In such an approach, a request message will carry the information of the nodes that it has traversed. When a node needs to forward a request message to a neighbor, the carried information will be used to prevent the message from being forwarded to a node that the message has traversed before. GNUnet adopts this approach, where the information of the nodes that a request message has traversed is carried in the message using a bloom filter [4].

The second approach aims to detect routing loops and react accordingly. Freenet adopts this approach. In Freenet, a unique identifier (UID) is carried by each request message and maintained by the nodes that have processed the message. When a request message with an old UID value arrives at a node, the node will send a failure message *Reject with Loop* to the upstream node n where the message comes from, so that node n will choose a different neighbor to forward the message to. A principal requirement of any loop handling scheme in apCSNs is that it should not leak any message forwarding information that can undermine the anonymity of the user who originates the message. However, both approaches leak certain level of message forwarding information, which can be exploited by attackers to undermine or compromise the user anonymity. For example, based on the bloom filter carried in a request message in GNUnet, an attacker can determine the set of nodes that have seen (either originated or forwarded) the message.

Similarly, reactive loop detection schemes such as the one adopted in Freenet can also be exploited to determine the set of the nodes that have seen a request message. For example, in order to determine if a node has seen a request message with a particular UID value before, an attacker can send a specially crafted probe message with the interested UID value to the node. The attacker can confirm that the node has seen the request message if a *Reject with Loop* failure message is returned. Moreover, as shown in [5], for a large portion of content request messages, the complete forwarding path can be determined, and the originating machine can be identified. More discussions on the impact of loop handling schemes on the user anonymity of apCSNs will be provided in Section II.

In this paper we will develop a new routing loop handling scheme named Reroute On Loop (ROL). In ROL, each request message will carry a UID value, and each node in the network will maintain the history of the UID values of the recent request messages that have traversed the node. In addition, for each UID value, the node will also record the set S of the neighbors to which the corresponding request message has been forwarded by the node and the neighbors from which the message has come. When a node n receives a request message m with an old UID value, node n will forward the message m to the next closest neighbor (based on the routing protocol of the apCSN), excluding the neighbors in set S. In this way an attacker cannot determine if a node has seen a request message before by sending a specially crafted message with an old UID value. An old message at node n will be rerouted to an unused neighbor (or discarded due to other properties of the request message), and critically, no failure message revealing the fact that node n has seen the message will be returned to the upstream node where the message comes from.

A critical concern of ROL is its performance impact on the forwarding of request messages on the resulting apCSN. Given that a request message may traverse a node multiple times, messages in an apCSN with ROL may traverse a longer path compared to the ones without ROL. Moreover, many apCSNs have a bound on the number of nodes that a request message can traverse, and therefore, ROL may limit the search scope of a request message. Consequently, a content insert message may not be able to identify the ideal location where the content should be inserted, and a content request message may not be able to reach the target location where the message should be routed to. In order to understand the efficacy and effectiveness of ROL, in this paper we will perform simulation studies using the Thynix simulator developed by the Freenet project [6], with a number of different network topologies, ranging from small-world topologies to random network topologies [7], [6].

Our simulation studies show that, compared to the current loop handling scheme in Freenet, ROL only has minor performance impacts on the lengths of message forwarding paths on various network topologies (and consequently the search scope of a request message if a message is bounded by the number of hops it can traverse). For example, the average routing path lengths of messages with ROL are only increased by less than 1 hop compared to the current loop handling scheme of Freenet on small-world network topologies. Our simulation results confirm that ROL is a practical scheme, and can be deployed on Freenet and similar apCSN systems.

The remainder of the paper is structured as follows. In Section II, we provide the necessary background on existing apCSNs and their routing schemes to illustrate the impact of loop handling on the user anonymity. In Section III we develop the new ROL scheme. We perform simulation studies in Section IV, and discuss related work in Section V. We conclude the paper in Section VI.

II. Background

In this section we provide the necessary background on the operations of two representative apCSNs, Freenet and GNUnet, including their formation of network topologies, their routing algorithms, and how they handle routing loops. Towards the end of this section, we will also briefly discuss the operations of another apCSN named OneSwarm. We refer interested readers to [1], [2], [3], [4], [8] for more details on these apCSNs.

A. Freenet

Freenet is a popular apCSN, with the objective to provide user anonymity for both content publishers and retrievers. In Freenet, each node (a machine running Freenet) contributes a portion of its hard disk space to form a global distributed storage sharing system. Each node in Freenet is associated with a location in the *circular* range [0, 1], where location 0 and location 1 are considered identical. The location of a node is randomly chosen by the node when it first joins Freenet.

Freenet nodes try to form a small-world network topology [7], where, with a high probability, the majority of neighbors of a node n have a location that is close to the location of node n. At the same time, a node may also connect to neighbors with a far-away location, which provide short-cut for routing messages to a remote target location. In default, each Freenet node can have up to 40 neighbors.

Freenet data insertion and retrieval involve a number of different types of messages. In this paper we will use *Content Hash Key* (CHK) based content request message as an example to illustrate the routing behavior of Freenet. Other request messages are handled in a similar fashion. The routing key of a CHK content request message is the SHA-256 hash of the corresponding data to be retrieved. The CHK routing key is used to uniquely identify the corresponding data on Freenet. To a degree, CHK messages are the most fundamental in Freenet. For routing purpose, the CHK routing key is converted into a location value in the same range of [0, 1], and the corresponding request message will be routed towards that location when received by a node.

A (CHK) content request message is issued by a node when the user requests a file on Freenet. (In this paper we use the two terms *file* and *content* interchangeably.) Each message is associated with a CHK routing key, a hop-to-live (HTL) value, and a unique identifier (UID). In essence, Freenet adopts a *greedy routing* scheme to route a request message towards the target location. When a node receives a content request message, it will check if it has the corresponding data in its local data store. If it does, it will return the data along the reverse path of the request message, and nodes along the path may cache the data to better serve future requests on the same file. If the current node does not have the requested data, it will forward the request message to the next closest neighbor based on the routing key.

The HTL value in a content request message is used to determine the number of hops the message can be forwarded along a forwarding path. Each intermediate node will decrease the value, and when it reaches 0, the corresponding request message will be discarded instead of being forwarded. In addition, a *Data not Found* failure message will be sent back to the upstream node, which will be further propagated back to the content requester to indicate the failure of the content request. For security reasons, the HTL value associated with a request message may not be decreased when it equals the maximum initial value (18 in default) or 1.

When a request message cannot be forwarded due to reasons other than HTL = 0 (for example, no additional neighbors are available), the request will be backtracked to the upstream node where it comes from, in the sense that the upstream node will forward the request onto the next closest neighbor (if it is available). This process continues until either HTL becomes 0, the requested data is found, or all possible routes have been tried but the data cannot be found.

The UID in a content request message is used by nodes to uniquely identify a message, and to detect routing loops. In Freenet, UIDs are randomly generated and are of length of 8 bytes, it is unlikely that two unrelated messages will have the same UID value in Freenet. We note that, although nodes in Freenet aim to form a small-world network topology, routing loops may be formed on Freenet due to a number of factors.

First, the small-world network topology is not as structured as the structured peer-to-peer (P2P) systems such as Chord [9]. In the structured P2P systems, deterministic routing can be used, and it can be guaranteed that routing loops will not be formed (at least in static cases). However, structured P2P systems can themselves leak too much message forwarding information [10], based on the network topologies and the routing protocols. The greedy routing used in small-world networks cannot guarantee that a request message is always forwarded to the ideal target location. The greedy routing protocol, to a degree, is only a best-effort approach based on the local information available at a forwarding node. As such, multiple tries may be carried out in forwarding a message, which increases the chance to form a routing loop. In addition, in order to deal with local minima and in an effort to locate the ideal target location of a message, some special techniques are also adopted (for example, forwarding a message to the next best neighbor, even if the next one is farther away from the target location of the message compared to the current node), which further increases the chance of routing loop formation.

Second, due to the nature of P2P systems, the network topology of Freenet is formed in a distributed fashion, and may not be an ideal small-world network, which further degrades the performance of the greedy routing and increases the chance to form routing loops. In order to detect routing loops, each node maintains the history of the messages that it has recently seen in the form of UID values. When a node receives a request message, it will first check if it has seen the corresponding UID before. If it does, a *Reject with Loop* failure message will be returned to the upstream neighbor where the message comes from. Otherwise, the message is processed according to the routing protocol of Freenet.

However, as shown in [5], the loop handling mechanism in Freenet can be exploited by an attacker to identify all the nodes that have seen a request message. Moreover, when the path traversed by a request message satisfies certain conditions, the complete forwarding path can be re-constructed and the originating machine of the message can be identified. One of the key insights utilized by the traceback attack in [5] is that, by observing the responding message from a node to a specially crafted probe message with an interested UID value, an attacker can infer whether or not the node has seen a concerned content request message with that UID value. In order to prevent the leakage of message forwarding information while detecting and responding to routing loops, we need a new loop handling scheme in Freenet.

B. GNUnet

GNUnet nodes form a Kademlia-like network topology [11], and message routing is carried out in two stages. In the first stage, a request message is routed randomly in the network. After traversing a sufficient number of hops (roughly $log(n)$, where n is the number of nodes in a GNUnet network), in the second stage, the request message is forwarded according to the Kademlia protocol, with an exception that, the routing is carried out in a recursive fashion instead of an iterative fashion as in the original Kademlia system, due to the anonymity requirement of GNUnet.

The rationale behind the random routing in the first stage is to make the lookup of a file independent of the location of the originating machine. Although it was not explicitly stated, we believe that the random routing also helps to improve the anonymity strength of GNUnet. We note that Kademlia is a structured network topology, should the set of nodes that have originated or forwarded a request message become known, the complete forwarding path of a message in Kademlia can be re-constructed. By including a random routing stage, an attacker can only trace a request message back to the last node involved in the random routing stage based on the routing protocol of Kademlia, but not the originating machine of the message. Therefore, the random routing stage helps improve the overall anonymity strength of GNUnet.

However, random routing also introduces a new problem into GNUnet. Due to random routing, loops can be formed in the forwarding of a request message. To prevent this problem, each request message in GNUnet carries the information of the nodes traversed by the message using a bloom filter. When a node needs to decide the next hop to forward a message to, the bloom filter carried in the message is used to exclude the nodes that have seen the message before. This approach has false positives, but will not have false negatives, which can guarantee the prevention of routing loops.

However, given the bloom filter is carried in the message, an attacker receiving the message can determine all the nodes that have seen the message before (it may mistake some nodes that have not seen the message before, but that probability should be very small, due to the objective of bloom filters used in GNUnet), which degrades the anonymity of GNUnet. Furthermore, after the set of all nodes that have seen a message is identified, in certain cases, a partial message forwarding path may be re-constructed for the nodes involved in the second routing stage, which further degrades the anonymity of GNUnet. Overall, loop prevention techniques based on information carried in a request message have some undesired implications on the anonymity strength of the resulting apCSNs, given that the information is readily available to an attacker who can observe the request message.

C. OneSwarm

The loop handling schemes adopted in Freenet and GNUnet are the representative ones used in existing apCSNs. OneSwarm adopts a slightly different loop handling scheme [8]. In OneSwarm, nodes form an unstructured, random network topology. And as such, a search message is flooded by a node to its neighbors (with certain restrictions), instead of being routed as in Freenet or GNUnet. In order to prevent a search message from being flooded more than one time at a node, each node maintains a set of rotating bloom filters to keep track of the search messages that have been recently flooded by the node. When an old search message arrives at a node, the message will not be further forwarded and no response message will be returned to the upstream node where the message comes from. This is different from the loop handling scheme used in GNUnet, where a bloom filter is carried in a request message.

We note that this scheme works in OneSwarm because of the message flooding mechanism used in OneSwarm. It will not work if messages are routed instead of being flooded. Due to scalability concerns of flooding, in this paper we will only focus on loop handling schemes that can work with message routing mechanisms instead of only message flooding mechanisms.

III. Reroute On Loop

The two loop handling schemes adopted by Freenet and GNUnet both leak certain level of message forwarding information that can be exploited by an attacker to compromise or undermine the user anonymity of these networks. It is critical to develop a secure loop handling scheme in order to improve the anonymity strength of these apCSNs. We note that loop prevention schemes such as the one adopted in GNUnet would require node traversal information to be carried in a message itself, which naturally leaks certain message forwarding information. In this paper we only consider loop detection and handling schemes, which do not have this requirement.

The loop handling scheme in Freenet leaks message forwarding information because a node will respond with a *Reject with Loop* failure message if it receives an old request message. A potential approach to addressing this problem is to design and use a general failure message, instead of using a failure-specific response message, as briefly discussed in [5]. In particular, when a forwarding loop is detected at a node, instead of sending a *Reject with Loop* failure message to indicate there is a forwarding loop, the node should send to the upstream neighbor a general failure message, so that the neighbor cannot infer the specific reason of the failure.

However, this approach has some important implications on the optimization and performance of Freenet, and more importantly, after careful examination, we note that it can still be exploited by an attacker. For example, by retrying a number of request messages with different UID values and contents, an attacker can determine a failure is caused by a specific UID value (routing loop) or failure of content lookup. As a consequence, an attacker can determine if a node has seen a request message before. Another approach is to not respond to an upstream neighbor any failure message at all when a forwarding loop is detected by a node (as in OneSwarm). However, without the failure message, the upstream neighbor cannot detect the routing problem and cannot forward the request message to a different node. Therefore, the critical issue is how to ensure that a request message encountering a forwarding loop can be routed continuously towards its target location.

In this section we will develop a new loop handling scheme, named Reroute on Loop (ROL) that will not leak any message forwarding information. In essence, ROL is very similar to the loop detection and handling scheme in Freenet, with a minor but critical difference. ROL can be adopted in many different apCSNs. However, in order to make our discussion more concrete, we present ROL in the framework of Freenet (note that ROL is only concerned with loop handling, other aspects of message routing are apCSN specific). In ROL, each request message is associated with an UID value, and each node maintains a history of the UID values of the recent request messages that the node has seen so as to detect routing loops. In addition, for each UID value, a node n will also record the set S of the neighbors to which the corresponding message has been forwarded *by node* n and the neighbors where the message came from.

When a request message arrives at a node n, the node will first check if it has seen the message before, based on the UID of the message. (In order to focus on message routing and loop handling, we assume node n does not have the content that the message is looking for. Otherwise, the content will be returned on the reverse path of the request message, and the message will not be further forwarded.) If it has not seen the message before, the message is forwarded according to the routing protocol of the apCSN. If node n has seen the message before, it will continue forwarding the message to the *next* closest neighbor, *excluding* the ones in set S. Importantly, no failure message will be returned to the upstream neighbor to indicate the detection of the routing loop. (A failure message may be returned later due to other reasons rather than routing loops, for example, data cannot be found or route cannot be found.)

ROL impacts on message path lengths and HTL. Note that a node n determines the next closest neighbor cn to which a message m should be forwarded only based on the local forwarding information available at node n. Therefore, node n may forward message m to a neighbor cn who has seen the message before, as long as node n has not used neighbor cn before (for message m). Consequently, a message may traverse a node multiple times in ROL. Figure 1 shows an example where a node is traversed 2 times by a content request message. In the figure, the numbers along the edges show the order of the message forwarding. In the figure, node A originates a content request message; it is forwarded to node B, and then to nodes C, D, E, and F, in that order. Node F then forwards the message back to node C, without knowing that node C has seen the message before. When node C receives this message from node F, it checks and notices that this is an old request message, it will then select the next best neighbor to forward the message to, excluding nodes B and F (from which node C received the message) and node D (to which node C has forwarded the message previously). In the example, node C select node G as the next hop to forward the message to.

Given that loops are admitted in ROL and a node along the forwarding path of a message may be traversed multiple times by the message, ROL may have a deteriorating impact on the performance of Freenet in terms of message path lengths, which is the key concern over the adoption of ROL in real-world apCSNs such as Freenet. In the next section,

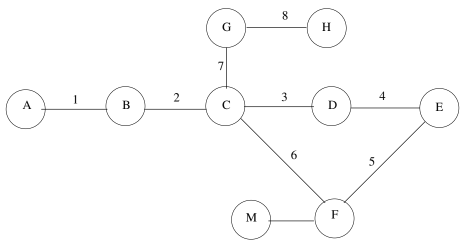

Fig. 1. Forwarding of a content request message.

we will carry out extensive simulation studies with various network topologies to investigate the performance of ROL. However, we first note that previous studies have shown that the probability to form a forwarding loop in small-world network topologies is small [12]. Therefore, ROL is rarely triggered in the normal forwarding of request messages (it is mainly used to prevent attackers from exploiting the loop handling scheme). In addition, we also make an observation here to show that the performance of ROL may not be as bad as first conceived.

In Freenet, when node C receives the request message from node F (see Figure 1), it will return a *Reject with Loop* failure message, so that node F will select another neighbor to forward the message to. Without loss of generality, let M be next hop to which the message will be forwarded by node F in *Freenet*. In contrast, in ROL, it is node C who will decide how the message should be further forwarded. Let $d(n)$ denote the distance from node n to the destination of a request message. We note that, under normal conditions, $d(C) < d(M)$, because node F selected node C over node M when it first decided to which node the message should be forwarded. Put in another way, node C is closer to the target location of the request message than node M. It is inconclusive to state which of the two nodes (C and M) is at a better position to forward the message to its target location, although some of the neighbors of node C have been used before, which decreases the search capability of node C.

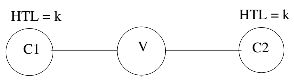

Fig. 2. Implication of HTL operation.

Recall that each request message is associated with an HTL value, which is used to prevent a message from looping forever in Freenet. HTL puts a constraint on the search scope of a request message. Given that a message may traverse a node n multiple times in ROL, it will further limit the search scope of a request message, if HTL is decreased each time the message passes node n. One naive solution to the problem would be that, a node will not decrease the HTL value of a message, if it has seen the message before (therefore, the HTL value is only decreased once when a node first receives the message). However, this solution may be exploited by an attacker to determine if a node has seen a message before. See Figure 2 for an example. In the figure, let us assume that an attacker wishes to determine if node V has seen a request message with a particular UID value. It can connect two attack nodes $C1$ and $C2$ to node V [13], [5], and then send a specially crafted request message to node V from node $C1$ with a particular HTL value, say k. If attack node $C2$ receives the request message from node V with an unchanged HTL value ($HTL = k$), the attacker can infer that node V has seen the request message before.

One way to address this issue is to let each node decrease the HTL value with a preconfigured probability (as done when HTL equals the maximum initial value or 1 in Freenet). In this way, an attacker cannot determine if an unchanged HTL is caused by the forwarding of an old message, or due to the probabilistic behavior of HTL manipulation. However, it has a side effect that the message search paths may become much longer, which may not be desirable as longer search paths will degrade the performance of Freenet.

As we will show in the next section, the performance impact of ROL on the message path lengths is minor on various network topologies. Given this observation, we will not change the behavior of HTL manipulation due to ROL. That is, each node will decide if it will decrease the value of HTL according to the original protocol of the corresponding apCSN.

How should content be transfered back to the requester? In ROL, temporary routing loops can be formed in the sense that a node may see and forward a request message multiple times. This presents a unique issue on how the requested content should be propagated back to the requester of the content. Let us again use the network topology in Figure 1 as an example. Recall that, a request message is originated at node A, and forwarded along the path $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow C \rightarrow G \rightarrow H$, and assume that node H has the content that is being requested. When the requested content is delivered back to the content requester, there are two different paths (in this example). One is the *original reverse path* $H \rightarrow G \rightarrow C \rightarrow F \rightarrow E \rightarrow D \rightarrow C \rightarrow B \rightarrow A$, another one is a *short-cut path* $H \rightarrow G \rightarrow C \rightarrow B \rightarrow A$. That is, when the content is propagated back to node C from node G, node C can either forward back the data onto the original reverse path (to node F), or directly forward back the data to the earliest neighbor from which node C received the corresponding request message (in this case, node B).

Both approaches have their advantages and limitations. Using the short-cut path will minimize the response time to a content request message, and likely improve the user experience on the resulting apCSN. However, it also has its own shortcomings. First, in order to remove the state maintained at the nodes not on the short-cut path, some kind of special messages should still be sent along the forwarding path for the nodes not on the short-cut path, which complicates the apCSN protocol. For example, a new *request cancel* message can be sent from node C to the next hop along the original forwarding path, that is D, if short-cut path is used to propagate the content back to node A. When a *request cancel* message is received by a node, the corresponding state related to the request message will be removed, and the cancel message is further forwarded along the forwarding path. When the *request cancel* message is forwarded back to node C from node F, node C can simply discard the cancel message.

Second, using a short-cut path also has performance implications on Freenet. In Freenet, aggressive content caching is used to improve the probability that data is located and returned in a timely manner. Specifically, when content is returned along the reverse path in Freenet, nodes along the path will cache the received content (with certain restrictions related to security). Using short-cut paths will reduce the chance of data caching in the network.

Propagating content along the original reverse path is the simplest, without any change to Freenet. However, it is certainly undesirable, given that content can be returned to the requester on the short-cut path to improve the user experience on Freenet. Another subtle issue is that, given that the message paths could be slightly longer in ROL compared to those in Freenet, more copies of content could be cached in the network. However, given each node only has limited cache (storage) space, spreading more copies of the same content could affect the availability of other content in the network.

In this paper, we propose a hybrid approach to propagate content back to its requester, where content is forwarded onto both the short-cut path and the original reverse path. In the hybrid approach, a node along the reverse path of a message will forward the data back to all the upstream neighbors (in particular, the one on the short-cut path). Instead of caching content with a probability of 1 (with some restriction related to security), content is only cached at a node with certain probability. In this way, the impact of longer message path on caching should be minimized. When a node receives the content again (maybe multiple times, depending on the number of upstream neighbors), it will simply discard the content.

As an example, when node C receives the content from node G, it will forward it back to node B on the short-cut path, in addition, it will also forward the content to node F on the reverse path. When node C receives the content from node D, it will simply discard the content. Note that, if there are multiple loops at node C, it will need to forward the content to the upstream neighbor in each loop.

What if a request message cannot be forwarded concerning ROL? In an extreme situation, it may occur that all the neighbors of a node n have been involved in the forwarding of a request message. When this happens, node n cannot further forward the message to any other nodes. Should a failure message (such as *Route not Found*) is returned immediately, the upstream neighbor can infer that the more likely cause of the failure is that the node n has seen the request message before, instead of other routing problems. As a consequence, an attacker can exploit this behavior to determine if node n has seen a request message before. However, we note that, given the large number of neighbors that a node can have (up to 40 in Freenet), this situation should rarely occur. Second, whenever this really happens, node n can delay the delivery of the failure message for certain amount of time (for example, average processing time for a message to traverse a few nodes) to prevent the upstream neighbor from inferring the specific reason of a failure.

IV. Performance Evaluation

In this section, we perform simulation studies to investigate the performance of ROL. We will first describe the simulation setup, and then we will provide and discuss the results of the simulation studies. (Additional studies are included in [14].)

A. Simulation Setup

The simulation studies are carried out using the Thynix simulator coming with the Freenet project [6]. Thynix is a simulator developed to investigate the Freenet behaviors including probe routing and path folding. It supports the routing of Freenet request messages in the sense that, given a pair of source and destination nodes in Freenet, it can determine the path that a request message will follow in the Freenet using the Freenet (greedy) routing protocol. However, in order to scale to large Freenet network topologies, it does not support functions such as file insertion, storage/caching, or retrieval. We extend the simulator to support ROL. To ease exposition, we refer to the current loop handling scheme in Freenet simply as Freenet. We note that ROL and Freenet only differ in the loop handling behavior, they are identical in all the aspects of the Freenet operation. In particular, both of them use the greedy routing in order to forward a request message to its destination.

In order to thoroughly investigate and understand the performance of ROL compared to Freenet, we consider a number of key network properties in the simulation design, including network size (number of nodes), node degree, and network topology. We consider network sizes with 2000, 4000, 8000, and 10000 nodes, and two node degrees of 8 and 16. A node degree specifies the maximum number of neighbors that a node can have in a network. In combination, we have 8 different sets of networks, (2000, 8), (4000, 8), (8000, 8), (10000, 8), (2000, 16), (4000, 16), (8000, 16), (10000, 16), in the format of (network size, node degree). We refer to them as S1 to S8, respectively. For simplicity, we also use S1 to S8 to refer to the set of simulation studies performed on the corresponding network. We note that the current Freenet has about 3000 to 4000 nodes simultaneously online on average.

In terms of network topology (how nodes are connected), we consider a number of different network topologies, including both small-world topologies and random topologies. In the following we describe how nodes are connected in different topologies. As in the real-world Freenet, each node in a network will be assigned with a location randomly selected in the circular space $[0, 1]$, where locations 0 and 1 are considered identical. In a small-world topology, two nodes are connected (becoming neighbors of each other) with a probability that is inversely proportional to the distance between the two nodes [6], [7]. In a random network topology, nodes are randomly connected, regardless their distance.

Nodes in the real-world Freenet attempt to form a small-world topology, but there is no guarantee that they can achieve this goal. The network topology of the real-world Freenet is more likely to be some variation between a small-world topology and a random topology. For this reason, we will also consider hybrid network topologies, where x of neighbors of a node are selected randomly, and the remaining neighbors of the node are selected according to the small-world criterion. We consider $x = 5\%, 10\%, 20\%$, and 30%, respectively. Furthermore, since we focus on the investigation of the performance of ROL, in all the simulation studies we set HTL to a large value (2000) so that with a high probability we can always find a path from any source node to any destination node.

For ROL and Freenet, we perform 8 sets of simulation studies, $S1$ to $S8$, with each set consisting of two groups of simulation studies. One group uses a small-world topology, another random topology. In each simulation study (with a fixed network topology), we randomly select two nodes in the network, we determine the route from the source node to the destination node (using ROL or Freenet), and then we record the routing path length in the number of nodes along the path. We perform 1000 simulation studies in each group of simulation studies (with randomly selected pairs of source and destination nodes in each simulation study), which simulates 1000 random content requests on the network. We use average routing path lengths in each group as an indicator of the performance of a loop handling scheme. In general, a shorter average routing path length is preferred.

TABLE I
PROPERTIES OF THE NETWORKS USED IN SIMULATIONS.

			Average network path length	
Set	Size	Degree	Small-world	Random
S1	2000	8	4.230	4.053
S2	4000	8	4.589	4.413
S3	8000	8	4.949	4.765
S4	10000	8	5.076	4.882
S5	2000	16	3.265	3.083
S6	4000	16	3.528	3.377
S7	8000	16	3.789	3.638
S8	10000	16	3.872	3.709

Table I summarizes the properties of the 8 sets of networks used in the simulation studies. In the table we also show the average network path length of the corresponding network topology. The average network path length is a graph property independent of the (greedy) routing used in an apCSN. It allows us to combine both network size and node degree into a single parameter of the network. In general, a large average network path length indicates that nodes in a network are more spread, and the topology likely has a larger network diameter. As we can see from the table, the average network path length of a network is strongly affected by the node degree. As the node degree increases, the average network path length becomes smaller for a fixed network size. On the other hand, given a fixed node degree, the average network path length becomes greater as we increase the network size. Both are intuitively sound. It is also interesting to note that random networks have a shorter average network path length compared to the corresponding small-world networks. This could be related to the fact that random networks have less restriction on connecting two nodes than small-world networks.

B. Simulation Results

In this subsection we present the results of the simulation studies. First we present the results on small-world topologies and on the random network topologies using the 8 sets of networks. Towards the end of the section we present the results of the simulation studies using hybrid network topologies.

1) Small-world and Random Network Topologies: Table II shows the average routing path lengths of ROL and Freenet in the 8 sets of networks. From the table we can see that, overall ROL only has a minor performance impact on the average routing path lengths compared to Freenet. In particular, the increment of average routing path lengths of ROL is negligible compared to that of Freenet in small-world networks. All the increments are greatly less than 1 in the small-world networks. Moreover, in certain cases (for $S3$, $S4$), ROL actually has a shorter average routing path compared to Freenet, and in some other cases ($S5$, $S6$, and $S7$), there is no change in the average routing path length between ROL and Freenet. For the last case, we have checked that there is no routing loops caused by ROL, and ROL and Freenet have the same message forwarding paths.

TABLE II
AVERAGE ROUTING PATH LENGTHS OF FREENET AND ROL.

	Small-world networks		Random networks	
Set	Freenet	ROL	Freenet	ROL
S1	7.411	7.553	43.448	45.678
S2	8.706	8.880	86.725	93.604
S3	9.910	9.839	156.186	188.381
S4	11.071	10.997	208.433	215.617
S5	4.790	4.790	11.736	12.424
S6	5.257	5.257	21.165	21.522
S7	5.733	5.733	42.062	46.118
S8	5.951	5.957	50.307	58.128

The performance of ROL is somewhat worse on random networks compared to small-world networks. However, we note that Freenet also works worse in random networks compared to small-world networks. Therefore, although ROL has a greater absolute increment in the average routing path length in random networks, the relative increment compared to Freenet is still relatively small. For example, ROL has no more than 10% increase in average routing path length for the majority of random networks.

In order to better illustrate the performance of ROL and Freenet with respect to the network properties, we show in Figures 3 and 4 the average routing path length as a function of the average network path length. From the figures we can see that as the average network path length increases, in general the average routing path length also increases (with a notable dip in the random networks in Figure 4). This is expected because a longer average network path means that the nodes in the network are spread, and the network diameter is likely larger. In general a message will traverse more nodes in order

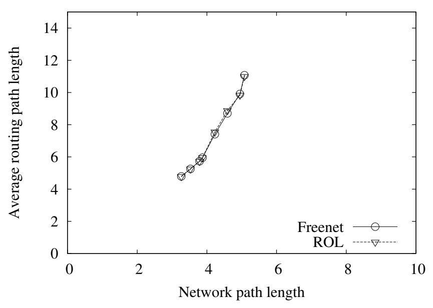

Fig. 3. Average routing path length (small-world networks).

to reach a destination in a more spread network for any routing algorithms.

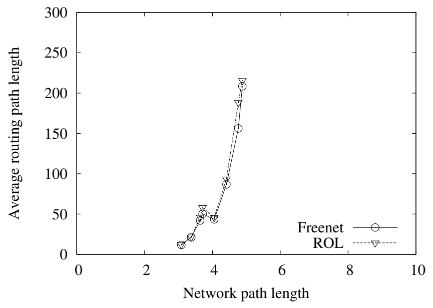

Fig. 4. Average routing path length (random networks)

Figures 5 and 6 show the empirical cumulative distribution function (CDF) of the routing path lengths for both ROL and Freenet, in small-world networks and random networks, respectively. To make the figures more legible, we use two networks ($S2$ and $S3$) as the representative examples. Data with other networks show a similar trend. From Figure 5 we can observe that, ROL and Freenet has a very similar CDF of routing path lengths, which again confirm that the impact of ROL on message path lengths should be very small compared to Freenet on small-world networks. In addition, both ROL and Freenet have relatively short routing path lengths, for example, more than 95% of messages have a routing path that is no greater than 18 hops, which is the default maximum initial value of HTL on Freenet.

In contrast, from Figure 6 we can see that both ROL and Freenet have much longer routing path lengths on random networks compared to small-world networks. (Note that the routing path length of 2000 of Freenet in $S3$ is caused by our limit on the HTL value in the simulation studies. The actual routing paths could be longer.) For an example, the majority of routing paths have a length that is greater than 50 hops, and a large number of routing paths have a length that is greater than 500 hops. Given that nodes in random networks are connected randomly, independent of their locations, we do not expect any routing algorithms, and greedy routing in particular, will work well in this type of networks. Despite the relatively large routing path lengths, we emphasize that ROL performs similarly as Freenet, put in another ways, ROL also

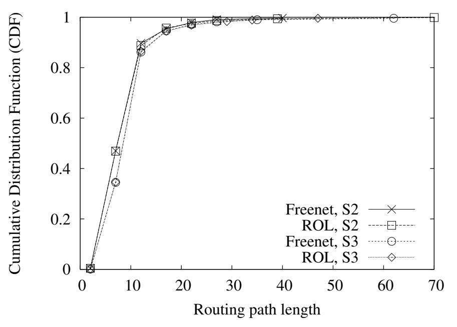

Fig. 5. Distribution of routing path lengths (small-world networks).

does not have major impact on routing path lengths compared to Freenet in random networks.

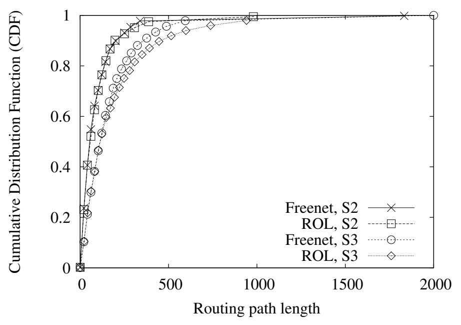

Fig. 6. Distribution of routing path lengths (random networks)

In order to better understand the impact of ROL on message forwarding, in Table III we show the number (and percentage) of messages that encounter a loop (traverse a node multiple times) during the forwarding from the source to its destination, for both small-world networks and random networks. Recall that in each group of simulation studies we perform 1000 content requests. From the table we can see that in small-world networks only a very small number of messages will encounter a loop in each group of simulation studies, ranging from 0 to 65, which is less than 7% of messages in all groups of simulation studies (with small-world networks). This shows that messages in small-world networks will rarely encounter loops with greedy routing, which has been observed in some previous studies [12]. The ROL in this context is mainly used to prevent attackers from exploiting the loop handling scheme.

In contrast, a large percentage of messages will encounter loops in all the groups of simulation studies on random networks. Again, given that nodes are randomly connected in random networks, it is not surprising that a message will be forwarded back to a node that has seen the message previously. We also observe that the node degree plays a key role in the formation of routing loops in both small-world networks and random networks. As the node degree increases (with a fixed network size), the probability for a message to encounter a routing loop becomes smaller. This is understandable; in an extreme case when a network becomes a clique (nodes have the largest degree), there will be no forwarding loops.

TABLE III
NUMBER OF MESSAGES IN LOOPS.

Set	# of messages in loops (%)	
	Small-world networks	Random networks
S1	35 (3.5%)	556 (55.6%)
S2	55 (5.5%)	664 (66.4%)
S3	45 (4.5%)	781 (78.1%)
S4	65 (6.5%)	789 (78.9%)
S5	1 (0.1%)	231 (23.1%)
S6	0 (0%)	377 (37.7%)
S7	0 (0%)	519 (51.9%)
S8	3 (0.3%)	586 (58.6%)

2) Hybrid Networks: Given that the network topology of the real-world Freenet is more likely to be a variation between a small-world network and a random network, in the following we perform simulation studies using hybrid networks. The hybrid networks are constructed using the parameters of $S3$, that is, the network size is 8000, and the node degree is 8 in these networks. Using other sets of networks will provide similar observation.

TABLE IV
AVERAGE ROUTING PATH LENGTHS ON HYBRID NETWORKS.

Network topology	Freenet	ROL
Small-world	9.910	9.839
5% random	10.373	11.036
10% random	10.947	12.091
20% random	15.201	15.152
30% random	18.015	23.611
(100%) random	156.186	188.381

Table IV shows the average routing path length on various hybrid networks, with $x = 5\%, 10\%, 20\%$, and 30%. For comparison, we also include the results for the small-world topology (0% random), and random topology (100% random). As we can see from the table, even with added randomness in networks, ROL can still perform well compared to Freenet, and in some case outperform Freenet, in terms of average routing path lengths. Overall the simulation studies confirm that ROL is a practical loop handling scheme, and can be deployed on apCSN systems such as Freenet, which aim to form a small-world network topology.

V. Related Work

In response to the traceback attack on Freenet [5], Ian Clarke has proposed the same idea of ROL [15]. We independently developed the ROL scheme, and critically, we carried out extensive simulation studies on the performance of ROL. As discussed in Section II, Freenet and GNUnet have their own loop handling schemes [2], [3]; however, they both leak certain level of message forwarding information that can be exploited to compromise or undermine the user anonymity of these networks. OneSwarm has a slightly different loop handling scheme [8]; however it only works in flooding based apCSNs instead of routing based apCSNs. ROL can work in routing based apCSNs.

Roos and Strufe proposed a family of routing algorithms named NextBestK [16], where a node n can choose up to K neighbors with the distance to the destination t worse than that from node n to t. ROL is close to an instance of NextBestK, with $K = \infty$ (or maximum number of neighbors a node can have). However NextBestK and ROL were developed for different purposes. While NextBestK was concerned with routing with a relaxed Kleinberg small-world network model [17], ROL was concerned with preventing the leakage of message forwarding information due to the handling of routing loops.

VI. Conclusion

In this paper we have developed a new loop handling scheme named Reroute-on-Loop (ROL) that would not leak any message forwarding information so as to improve the anonymity strength of the resulting apCSN. Using the Thynix simulator coming with the Freenet project we have also shown that overall ROL only has minor performance impacts on routing path lengths compared to Freenet with various network topologies. Our simulation studies confirmed that ROL is a practical loop handling scheme that can be deployed on apCSN systems such as Freenet.

References

[1] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proc. International Workshop On Designing Privacy Enhancing Technologies: Design Issues In Anonymity And Unobservability*, 2001.

[2] Freenet, https://freenetproject.org/.

[3] GNUnet, https://gnunet.org/.

[4] N. S. Evans and C. Grothoff, "R^5N : Randomized recursive routing for restricted-route networks," in *Proc. 5th International Conference on Network and System Security (NSS 2011)*, Milan, Italy, Sep. 2011.

[5] G. Tian, Z. Duan, T. Baumeister, and Y. Dong, "A traceback attack on Freenet," in *Proc. IEEE INFOCOM*, Turin, Italy, Apr. 2013.

[6] Freenet, "Simulator," https://wiki.freenetproject.org/Simulator.

[7] O. Sandberg, "Distributed routing in small-world networks," in *Proceedings of 8th Workshop on Algorithm Engineering and Experiments*, Jan. 2006.

[8] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Privacy-preserving P2P data sharing with OneSwarm," in *Proc. ACM SIGCOMM*, 2010.

[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 protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[10] G. Ciaccio, "Improving sender anonymity in a structured overlay with imprecise routing," in *Proc. International Conference on Privacy Enhancing Technologies*, 2006, pp. 190–207.

[11] P. Maymounkov and D. Mazires, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proc. First International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[12] G. Tian, Z. Duan, T. Baumeister, and Y. Dong, "Thrawting traceback attack on Freenet," in *Proc. IEEE GLOBECOM*, Atlanta, USA, Dec. 2013.

[13] T. Baumeister, Y. Dong, Z. Duan, and G. Tian, "A routing table insertion attack on Freenet," in *Proceedings of ASE International Conference on Cyber Security*, Washington D.C., USA, Dec. 2012.

[14] G. Tian, Z. Duan, T. Baumeister, and Y. Dong, "Reroute on loop in anonymous peer-to-peer content sharing networks," Department of Computer Science, Florida State University, Tech. Rep. TR-140715, Jul. 2014.

[15] Toad, "Consider reroute-on-loop," https://bugs.freenetproject.org/view.php?id=5467.

[16] S. Roos and T. Strufe, "Provable polylog routing for darknets," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*. IEEE, 2012, pp. 140–146.

[17] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM, 2000, pp. 163–170.