Thwarting Traceback Attack on Freenet

Guanyu Tian, Zhenhai Duan
Florida State University
{*tian, duan*}*@cs.fsu.edu*

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

***Abstract*—A traceback attack was recently developed on Freenet, which can identify the originating machine of a content request message, even if a single content request message has been issued by a content retriever. The traceback attack exploited a few fine-grained design and development decisions made in Freenet, including the unique identifier (UID) based mechanism to prevent routing loops of content request messages. In this paper we develop a simple yet effective scheme named dynID to thwart the traceback attack on Freenet. In dynID, the UID associated with a content request message is dynamically changed at the beginning portion of the message forwarding path. As a consequence, an attacker can only trace back a content request message to the node where the UID value is last changed; it cannot uniquely determine the originating machine of the message. Importantly, dynID only has negligible impacts on the performance of Freenet in locating content on the network. For example, our simulation studies based on the original Freenet source code show that, for all content requests, we can successfully locate the corresponding requested content.**

I. Introduction

Freenet is a popular anonymous peer-to-peer content sharing network, with the design objective to provide the anonymity of both content publishers and retrievers [2], [3]. Despite the time-proven high-level security mechanisms and algorithms adopted in Freenet, a traceback attack has been recently developed that can break the anonymity of content retrievers on Freenet [5]. More specifically, the traceback attack can identify the originating machine of a content request message, even if a single content request message has been issued by the content retriever.

In the traceback attack, after a monitoring node of an attacker observes a content request message, nodes controlled by the attacker will iteratively connect to suspect nodes, and query them to determine if they have seen the message previously, based on the reply message from a suspect node. The traceback attack exploited a few fine-grained design and development decisions made in Freenet. A key Freenet feature utilized by the traceback attack in determining if a node has seen a message is the unique identifier (UID) associated with each content request message.

The UID carried in a content request message is used to prevent routing loops of the message. In Freenet, each node maintains a set of UIDs carried by the messages that it has seen, i.e., it has either originated or forwarded. When a request message with an old UID arrives at a node, the node will discard the message and reply to the upstream node with a *Reject with Loop* failure message. The traceback attack exploited this feature by sending to a suspect node a specially crafted probe message with the same UID as that of the interested content request message, in order to determine if the node has seen the content request message previously.

In this paper we develop a simple yet effective scheme named dynID (dynamic UID) to thwart the traceback attack on Freenet. In dynID, the UID of a content request message will be dynamically changed at the beginning portion of the message forwarding path. Let n denote the node where the UID of a content request message is last changed along the message forwarding path. In dynID, an attacker can only trace back the message to n; it cannot uniquely determine if n or any node is the originating machine of the message.

A key concern over dynID is that, given that a content request message is not uniquely associated with a UID anymore, loops may be formed along the message forwarding path, which may limit the search scope of a content request message and result in the failure in locating the corresponding content on Freenet. We note that a number of design decisions of dynID and the Freenet make this unlikely, and dynID should only have negligible impacts on the performance of Freenet in locating content on the network. Our simulation studies using the original Freenet source code, extended with dynID, show that, for all the content requests, we are able to successfully locate the corresponding content, despite the existence of short-lived loops along the forwarding path of a content request message (in some of the simulation studies).

In this paper we present the details of the dynID scheme in thwarting the traceback attack on Freenet, and conduct simulation studies to investigate its performance impacts on Freenet in locating content on the network. The remainder of the paper is organized as follows. In Section II we provide the essential background on the traceback attack (and Freenet). In Section III we detail the proposed dynID scheme in thwarting the traceback attack on Freenet. We perform simulation studies on the impacts of dynID on the performance of Freenet in Section IV. We briefly discuss related work in Section V and conclude the paper in Section VI.

II. Background

In this section we provide the necessary background on the traceback attack on Freenet, and the critical Freenet features that were exploited by the traceback attack. We refer interested readers to [5] and [2] for the details of the traceback attack and Freenet, respectively.

This work was supported in part by NSF grants CNS-1041677, CNS-1041739, CNS-1127875, and CNS-1219579.

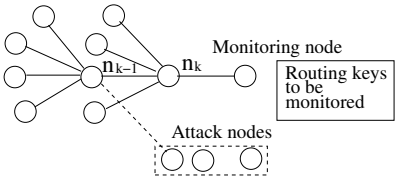

Fig. 1. Illustration of the traceback attack.

Figure 1 illustrates the basic structure of the traceback attack. In the traceback attack, an attacker will deploy a number of monitoring nodes in Freenet, with each maintaining a set of interested routing keys to be monitored. (A routing key is calculated based on the content to be requested and carried in a content request message.) A monitoring node will passively observe the content request messages passing through the node and try to match their routing keys with the routing keys to be monitored.

When an interested content request message is identified by a monitoring node m, it will forward a few pieces of information to an attack node controlled by the attacker, including the set of suspect nodes, which are the nodes who may have seen (i.e., either originated or forwarded) the message. Let $n \rightarrow m$ denote that a message is forwarded from node n to node m, that is, node m received the interested message from node n. Then the suspect nodes are the neighbors of node n (excluding node m since we know that m has received the message). The neighbor information of node n is available at node m due to the two-hop lookup algorithm employed by Freenet [5].

After an attack node receives the set of suspect nodes, it will connect to (i.e., become a neighbor of) each of the suspect nodes one by one. This component of the attack is carried out by exploiting the Freenet features on supporting an arbitrary node to join Freenet and the neighbor update mechanism at a Freenet node [1] and [5]. After an attack node is connected to a suspect node, it will send a specially crafted probe message to the suspect node to determine if the node has seen the corresponding content request message previously. This process exploited the Freenet feature on preventing routing loops of content request messages.

In Freenet, each content request message is associated with a unique identifier (UID), and each node also maintains the set of UIDs that it has seen. UIDs maintained by a node is classified into two categories. The first category is a list of UIDs associated with active request messages (whose corresponding reply message has not come back from the downstream node), and the second a queue of UIDs of the completed request messages (whose corresponding reply message has come back from the downstream node). When a request message arrives at a node, the node will first check if it has seen the UID carried in the message (in the set of UIDs of either category). If the node has seen the UID previously, it will immediately send back a failure message *Reject with Loop* to the upstream node where the message comes from. Otherwise, it will process the request message normally.

Based on the reply from a suspect node, the attack node can determine if the suspect node has seen the message previously. After a node n that has seen the message is identified, its neighbors will be in turn considered as suspect nodes and will be contacted and probed by an attack node to determine if any of them has seen the message previously. Similarly, the neighbor information of node n is available at the attack node because of the two-hop lookup algorithm of Freenet. This process continues until no new suspect nodes are identified. At the end of this process, we have the set of all nodes that have either originated or forwarded the content request message.

A set of lemmas were developed in [5] to help determine the originating machine of a content request message, after all nodes that have seen the message have been identified, by exploiting the routing algorithms employed by Freenet to forward a request message. In essence, the originating machine of a content request message can be uniquely determined, if the forwarding path of the message satisfies certain conditions defined in the lemmas. The experimental studies performed in [5] showed that, the originating machines of 24% to 43% of content request messages can be uniquely determined.

III. DynID to Thwart Traceback Attack

In this section we provide the details of the dynID scheme in thwarting the traceback attack on Freenet. We will present the intuition and the design choices of the scheme, and illustrate how it can help thwart the traceback attack on Freenet.

As we have discussed above, Freenet relies on a UID based mechanism to prevent routing loops of request messages, and this mechanism is one of the key features being exploited by the traceback attack. By iteratively connecting to and probing suspect nodes starting from a monitoring node where an interested content request message is observed, an attacker can identify all the nodes that have seen the message (based on the UID value). The originating machine of the message can then be determined if the message forwarding path satisfies certain conditions [5].

Given that the UID associated with a content request message plays a critical role in guiding the progress of a traceback attack, one way to thwart the traceback attack is to dynamically change the UID value associated with the message along the message forwarding path, which is the basic idea of the dynID scheme. In designing dynID, we must take into consideration a few factors that may affect the performance of Freenet, and whether or not an attacker can infer if the UID value of a message has been changed. In particular, we note that UID associated with a request message is used by Freenet to prevent routing loops. If a routing loop is formed along the forwarding path of a content request message, Freenet may not be able to locate the desired content even if the content exists on the network. Therefore, dynID should be designed in a way that can reduce the likelihood of forming routing loops, which is strongly related to when the UID value should be changed. In the following we will discuss these design choices of dynID.

We first provide some necessary Freenet background on controlling how far a request message can traverse in the network. In Freenet, in addition to UID and routing key,

a content request message is also associated with an HTL (hop to live) value, which is used to control how many hops (nodes) the message can traverse. The HTL field is initialized to a maximum value at the originating machine, and reduced by one at each node. When the HTL value becomes 0 at a node, the message is discarded by the node, and a *Data not Found* failure message is returned to the upstream node (and propagated back to the originating machine).

In Freenet, for security reasons, a node may not decrease the HTL value by 1 when the HTL value is the maximum value, so that a receiving node cannot infer if the upstream neighbor is the originating machine of the message, even if HTL has a value that is 1 less than the maximum value. Instead, the HTL value is only decreased by 1 with a preconfigured probability (default probability is 50%) when it equals the maximum value. Similarly, the HTL value may not be decreased to 0 when it already reaches 1. As a consequence of this behavior, a message may traverse a path that is longer than the initial (maximum) HTL value.

Location where UID value should be changed. Now let us discuss when the UID value of a request message should be changed. To simplify the discussion, we first assume that the UID is only changed once along the forwarding path. As mentioned above, a consequence of dynamically changing the UID value of a message is that routing loops may be formed along the forwarding path, and more importantly, the desired content may not be located because of routing loops. Figure 2 illustrates an instance where a routing loop is formed along the forwarding message.

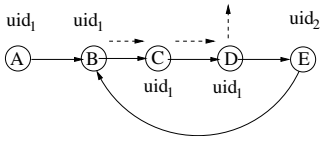

Fig. 2. A forwarding path with loop.

In the figure, node A originates a message with UID value of uid_1, which is forwarded along the solid line from A to B, and then to C, D, and E, in that order. Assume the UID value is changed at node E from uid_1 to uid_2, and the message is forwarded back to node B with the new UID value. Given that the incoming message carries a new UID value, node B considers the message a new message, and (likely) the message will be forwarded along the dashed line to nodes C and D in that order. Node D may again forward the message to node E. However, given that node E has seen uid_2 previously, it will directly reject the message, and node D will choose a different node to forward the message to (if it has one). We refer to the number of nodes involved in a loop (i.e., the nodes that forward the same message more than one time) as the size of the loop, and denote it as $|L|$. In Figure 2, $|L| = 3$.

We note that the larger the value of $|L|$ is, the greater impact a loop will have on the decrement of the HTL value, and consequently, the search scope of a content request message. (Note that the HTL value may not always be decreased by 1 at a node.) In the worst case, the HTL value could be reduced by $|L|$, that is, each node in the loop will decrease HTL by 1. If the value of $|L|$ is large, this can greatly affect the search scope of a content request message. For example, assume the maximum value of HTL is 18 (default HTL maximum value in Freenet), and further assume $|L|$ to be 15, the concerned content request message can only roughly visit 3 nodes that are not part of the loop along a forwarding path, and the likelihood for the content request message to locate the desired content will be small. Therefore, changing the UID value after a request message has traversed a large number of nodes may not be desired, due to the impacts on the performance of Freenet in locating requested content on the network.

Another related issue is if the UID value of a request message should be changed at a fixed (relative) location along the forwarding path, or each node should independently decide whether or not it will change the UID value. In order to illustrate the issue, we present a failed solution that we first worked on. In this failed solution, we choose to change the UID value after a message has traversed at least two nodes. More formally, let max denote the default maximum HTL value (18 in Freenet), and let htl_i denote the HTL value carried in a content request message when it arrives at a node n, and htl_o the new HTL value (may be $htl_i - 1$) to be carried by the outgoing message. A new random UID value will be assigned to the message by node n if $htl_o = max - 2$ and $htl_i > max - 2$.

We note that the two conditions ensure that the UID value is only changed one time along the forwarding path (when $htl_o = 16$). We also note that the UID value may be changed at node that is more than 2 hops away from the originating machine, given that the HTL value may not be decreased when it equals the maximum value. However, this solution cannot completely prevent the traceback attack. For example, if a monitoring node observes a content request message with $UID = 17$, it can safely infer that the UID value of the message has not been changed, and therefore, it can carry out the traceback attack to determine the originating machine of the message. Based on this observation, we conclude that changing the UID value at a fixed location (relative to the HTL value) cannot provide the desired anonymity strength. Instead, we need to let each node to independently decide if the UID value should be changed.

In order to balance the two requirements (the UID value should not be changed after a content request message has traversed a large number of nodes, and each node should independently decide if the UID value should be changed), dynID will only dynamically change the UID value of a message at the beginning portion of the message forwarding path. More specifically, the UID value will only be changed with a preconfigured probability at the nodes where the HTL of a message still equals the maximum value (i.e., $htl = max$). Recall that the HTL value of a message may not be decreased by 1 when it equals the maximum value, therefore, multiple nodes may independently change the UID value of the message. In essence, the basic process of changing UID in dynID

is similar to that of updating the HTL values in the original Freenet. If the UID value is changed at a node n, node n will also maintain the mapping between the two UIDs so that a reply message can be properly processed and propagated back to the upstream node.

Given this design of the dynID scheme, a natural question is if the UID value should also be changed at other parts of the forwarding path. We note that the objective of this work is to thwart the traceback attack that can *uniquely* identify the originating machine of a content request message. In dynID, an attacker cannot *deterministically* infer if the UID value has been changed at the beginning portion of the forwarding path. Therefore, it cannot uniquely determine if the last node being traced back to is the originating machine. Consequently, it is not necessary to change the UID value again at other parts of the forwarding path.

It is also worth noting that the UID value of a request message may not be changed along the forwarding path, depending on the length of the forwarding path, the dynamics of the HTL value of the message, and the probabilistic behavior of each node in deciding if the UID value should be changed. The desired content may be located close to the originating machine of the message (before the UID value is changed), or all the nodes where the HTL value equals the maximum value decide not to change the UID value. A related question is whether or not we should guarantee that the UID value should be changed at least one time along the forwarding path. By the same argument as above, even if $htl = max$ at the monitoring node, an attacker still cannot *deterministically* infer how many nodes the message has traversed or if the UID value of a message has been changed. Therefore, we do not need to guarantee that the UID value should be changed at least one time.

Based on the above discussions, in this paper the UID value of a content request message may only be changed at the beginning portion of the message forwarding path while $htl = max$, for the purpose of thwarting the traceback attack that can uniquely identify the originating machine of a content request message. We leave the investigation of probabilistic identification of the originating machine of a request message, and further changing the UID value along the message forwarding path as future work.

How secure is dynID? In dynID, an attacker can trace back a message from a monitoring node to the node n where the UID value is last changed along the forwarding path of the message. However, based on the information carried in the request message alone, an attacker cannot determine if node n is the originating machine or it is merely a node where the UID value is changed. (An attacker can compute the probability that node n is the originating machine, but it cannot uniquely determine if it is so.)

One complicating factor is the availability of the routing tables at the nodes in Freenet. Due to the two-hop message lookup mechanism adopted in Freenet, the routing table of any node in the Freenet can be obtained by an attacker (see [1] and [5]). At the high level, Freenet adopts a two-hop shortest-distance routing mechanism. Consider a message msg, and let $d(k)$ denote the distance from node k to the routing key carried in the message msg (based on the virtual location of node k and its neighbors). Let m denote a node on which the message msg arrives, and let N denote the set of all neighbors of node m. In the two-hop shortest-distance routing mechanism of Freenet, node m will choose the neighbor with the smallest distance to the routing key of the message msg, that is $argmin_k\{d(k)\}$.

In [5], a few lemmas were developed to help determine the originating machine of a message, by exploring the routing algorithm used in Freenet. A natural question is that if the routing algorithm (and the availability of routing tables of all nodes) can help an attacker to determine the node n where the UID is last changed is indeed the originating machine. To ease exposition, let us assume that node k in Figure 3 is the last node that an traceback attack have identified to have seen an interested UID value uid (of an interested request message msg), all the neighbors of node k did not see uid (except node n, from which the attacker traced the message back to node k). Without loss of generality, we further assume that node k is the source node of message msg among all the nodes that the traceback attack has identified to have seen the corresponding UID value. That is, the message is first forwarded to node k, before being further forwarded into any other nodes being identified by the traceback attack, based on the UID value observed by the monitoring node.

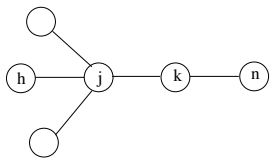

Fig. 3. Can node j forward a message to node k if node h is more preferred?

Let $N(k)$ denote the set of neighbors of node k (excluding node n for the simplicity of discussion). If the attacker can conclude based on the routing tables of $N(k)$ and the routing algorithm in Freenet that none of the nodes in $N(k)$ can have forwarded the message to node k, then it can also conclude that node k is the originating machine of the message (recall that we assume node k is the source node of all nodes that have been identified by the traceback attack based on the UID value).

In the following we show that, although the routing tables of all nodes and the specifics of the routing algorithm of Freenet are available to an attacker, it cannot determine if a neighbor of node k can have forwarded a message to the node. We note that despite the similarity between the problem being faced here and the problem being faced in [5], the problem faced here is much harder. In [5], an attacker knows all the nodes that have seen a particular UID value, it only needs to determine the direction of a message forwarding in order to determine if a machine is the originator of a message. In contrast, an attacker in the dynID scheme does not know all the nodes that have either originated or forwarded a message.

One potential approach an attacker may try to determine if node k is the originating machine is to check if it is possible for any of the neighbors $j \in N(k)$ to forward the message msg to node k, based on the routing tables of the neighbors. If it is impossible for any of the neighbors to forward the message to node k, then the attacker can conclude that node k is the originating machine of the message. Following the methodology in [5], it is enticing to believe that if a neighbor j has a more preferred neighbor h than node k (i.e., $d(h) < d(k)$), node j will not forward the message to node k, and if this applies to all neighbors of node k, none of them could have forwarded the message to node k. However, this observation is incorrect. Even if $d(h) < d(k)$ holds, it is still possible for node j to forward the message to node k.

One possible scenario is that node j tried to forward the message to node h first (because it is more preferred); however, the desired content is not located along that path, and the message is backtracked to node j, which then chosen node k. Another possible scenario is that, the message was actually forwarded by node h to node j, and then forwarded by node j to node k. There are other scenarios where it is possible for node j to forward the message to node k, even if more preferred neighbors of node j exist. Given that it is alway possible to make a case that the message may have been forwarded by a neighbor to node k, the attacker cannot conclude node k is the originating machine, even if it is the source node among all the nodes that have identified by the traceback attack to have seen a request message.

Per node or per message probability to change the UID value. In the original Freenet, the HTL value is kept the maximum value on a per node basis. More specifically, the behavior of a node to decrease or not to decrease the HTL value when $htl = max$ is determined when the node is started (joining Freenet). After the behavior is determined, it will be applied to all content request messages in the same manner. However, it provides a security hole in the sense that an attacker can easily determine the behavior of node in terms of the HTL update. For example, an attacker can connect two nodes to a target node, and send a content request message from one attack node to request content stored on another attack node. By observing the HTL value at the receiving attack node, it can infer if the target node will decrease HTL value or not (when it equals the maximum value).

Similarly, using the same method, an attacker can determine if a target node will change the UID value (when $htl = max$), should changing UID value be performed on a per node basis. Moreover, an attacker can profile all nodes on Freenet beforehand, instead of during a traceback attack. Profiling the behavior of all benign nodes on Freenet (in terms of HTL and UID update) will help an attacker in determining the originating machine of a message, at least in some special cases. For example, assume that node k is determined as the source node of all nodes that have been identified by a traceback attack, and assume (in an extreme case) that all neighbors of node k will decrease the HTL value (when $htl = max$), then the attacker can safely infer that node k is the originating machine if, for example, the observed HTL value equals the maximum value. For this reason, in dynID, both the HTL value and the UID value are updated on a per message basis instead of on a per node basis. That is, a node will determine if the HTL value and UID value of a message should be updated, independent from other messages that have been forwarded by the node.

IV. Performance Evaluation

In this section we conduct simulation studies to investigate the impacts of the dynID scheme on the performance of Freenet in locating content on the network. We first describe the set-up of the simulation studies, and then we present the performance results.

A. Simulation Set-up

The performance studies are carried out using the simulator coming with the Freenet project. Unlike other simulators that re-implement (and normally simplify) a product system, the simulator coming with Freenet uses the original Freenet source code (version 0.7, the latest version of Freenet). Put in another way, the behavior of nodes in the simulator is identical to that of nodes in the real-world public Freenet. We extend the source code of Freenet to support the dynID scheme.

All the Freenet networks we used in the simulation studies consist of 400 nodes, and each node can have up to 6 neighbors. The networks are Kleinberg (small-world) networks and constructed in the following manner [4]. As in the real-world Freenet, each node in a network will be assigned with a location in the circular space $[0, 1]$, where location 0 and 1 are considered identical. The set of locations used are evenly distributed in the circular space, with a distance of $1/N$, where N is the number of nodes in the network (400). The first location is 0, the second 0.0025, the third 0.0050, and so on.

Next we describe how a node selects neighbors. A node n will select half of the maximum number of neighbors based on a probability that is reversely proportional to the distance between node n and a candidate node. After all nodes have selected half of the maximum number of neighbors (i.e., 3 in our case), node n may have between 3 and 6 neighbors (node n may be selected by other nodes as a neighbors, therefore, it may have more than 3 neighbors). For each node n with $b < 6$ neighbors, we randomly select $6 - b$ nodes as its neighbors among all the nodes who still need more neighbors.

We run 10 sets of simulation studies, with each set containing 100 simulation runs. Each set of simulation studies use the same network topology (i.e., 10 different network topologies are used). In each set of simulation studies, we insert 100 different files at randomly selected nodes, and then retrieve them one by one from randomly selected nodes. We refer to the 10 sets of simulation studies as $S1$ to $S10$, respectively. In all simulation studies, we use the Freenet default values unless otherwise stated. In particular, we use the default HTL value (18), and the default probability 50% to decrease HTL when it has the maximum value. The probability to change the UID value in dynID when $htl = max$ is also 50%.

B. Results

In this section we present the results of the simulation studies; we focus on the impacts of the dynID scheme on the performance of Freenet in locating content on the network. As we have discussed early, loops may be formed along a message forwarding path in dynID, which will (likely) decrease the HTL value and limit the search scope of a content request message. As a consequence, we may not be able to return the desired content for a content request message, if the loop situation becomes very bad (which has been taken into consideration in the design of dynID).

TABLE I
RESULTS OF SIMULATION STUDIES.

Set	Total	Successful
S1-S10	1000	1000

As shown in Table I, for all the 1000 content requests, we are able to successfully locate the desired content. A number of factors should have contributed to the desired performance of dynID. First, dynID limits the change of the UID values only to the beginning portion of a message forwarding path. This design coupled with the probabilistic behavior of decreasing HTL values implies that the size of a loop, even if one is formed, will not be large. (The number of nodes with $htl = max$ along a message forwarding path follows a geometric distribution with $p = 0.5$.)

Moreover, the two-hop lookup algorithm and the semi-structured network topology also help to route a content request message towards the right direction [5]. In addition, a large initial value of HTL (18) also reduces the chance for a content request message to back track (to the originating machine). Combining all these factors, the chance for a forwarding path to enter a loop caused by dynID is small. In addition, even if a loop is formed, its impact on affecting HTL (and therefore limiting the search scope of the request message) should also be small (see below).

In order to better understand the results of the simulation studies, in Table II we show the number of message forwarding paths of various properties. A forwarding path is considered linear if every node n on the path only forwards the message to a single downstream node one time. Consequently, paths containing loops caused by dynID (*loop* in the table) and paths where *Reject with Loop* occurs (*Rejected loop* in the table) are non-linear paths. In the table the column *L & R* shows the number of paths that contains both loops and Rejected loops. As we can see from the table, the majority of message forwarding paths are linear. Moreover, we only have one loop that is caused by dynID, and the size of the loop is very small (2), which should only impose a minimal impact on the search scope of a content request message. Note that $S8$ contains more than 100 forwarding paths. When a *Data not Found* failure (because $htl = 0$) is returned to the originating machine, it will automatically start a new content request message with a different UID value in the original Freenet [3]. We consider the path followed by this new content request message as a separate path from the previous request message.

TABLE II
PROPERTIES OF MESSAGE FORWARDING PATHS.

Set	Linear	Non-linear			
		Total	Loop (size)	Rejected loop	L & R
S1	99	1	0	1	0
S2	100	0	0	0	0
S3	99	1	0	1	0
S4	98	2	0	2	0
S5	99	1	0	1	0
S6	99	1	0	1	0
S7	100	0	0	0	0
S8	99	2	1 (2)	2	1
S9	100	0	0	0	0
S10	98	2	0	2	0

V. RELATED WORK

Shortly after the traceback attack was identified on Freenet, the Freenet project developed a quick patch to mitigate the traceback attack, by removing the queue of UIDs associated with the completed request messages [6]. This scheme limits the flexibility of the traceback attack in terms of the time window that a traceback attack can be carried out. However, given that the list of UIDs associated with active request messages are still maintained by each node, the traceback attack can be still carried out, albeit with a smaller time window due to the scheme. The lightweight dynID scheme can more effectively thwart the traceback attack on Freenet. In particular, an attacker can only trace back a content request message to the node where the UID value is last changed; it cannot uniquely determine the originating machine of the message. It is worth noting that the two schemes complement each other, and can be combined to provide stronger user anonymity for content retrievers on Freenet.

VI. CONCLUSION

In this paper we have developed a simple yet effective scheme named dynID to thwart the traceback attack on Freenet. In dynID, an attacker can only trace back a content request message to the node where the UID value is last changed; it cannot uniquely determine the originating machine of the message. Importantly, the developed dynID scheme only has negligible impacts on the performance of Freenet in locating content on the network.

REFERENCES

[1] 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.

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

[3] Freenet. https://freenetproject.org/.

[4] Freenet. Kleinberg networks. https://wiki.freenetproject.org/Kleinberg_network/.

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

[6] Toad. How safe is Freenet anyway? http://amphibian.dyndns.org/flogmirror/#20120911-security.