SurePath: An Approach to Resilient Anonymous Routing

Yingwu Zhu[1] and Yiming Hu[2]
(Corresponding author: Yingwu Zhu)

Department of CSSE, Seattle University[1]
901 12th Ave., P.O.Box 222000, Seattle, WA 98122-1090 (Email: zhuy@seattleu.edu)
Department of ECECS, University of Cincinnati, Cincinnati, OH 45221-0030[2]

(Received Mar. 21, 2006; revised and accepted May 31, 2006)

Abstract

Current anonymous systems either use a small, fixed core set of mixes or randomly choose a sequence of peer nodes to form anonymous paths/tunnels. The resulting paths are fragile and shortlived — that is, a path fails if one of the mixes/nodes fails or leaves the system. In this paper, we propose SurePath, a failure-resilient tunnelling approach for anonymity built on distributed hash tables (DHTs). SurePath aims to make anonymous tunnels fault-tolerant to node failures. The basic idea behind SurePath is to decouple anonymous tunnels from "fixed" nodes and form anonymous tunnels from dynamic mix nodes by relying on DHT routing infrastructure and data replication mechanism. We have implemented SurePath in Java on FreePastry 1.3. We also provide analysis of anonymity and performance evaluation for SurePath.

Keywords: Anonymity, forward tunnel, peer-to-peer, relay set, reply tunnel

1 Introduction

Many Internet applications such as anonymous web-browsing, anonymous e-mail services and private P2P file sharing, need anonymization to provide anonymity for the participants such that their identities cannot be revealed by third-party observers. There are different types of anonymity [12]. *Initiator anonymity* hides the identity of the initiator from all other nodes including the responder. *Responder anonymity* means that the identity of the responder is hidden to all other nodes including the initiator. *Mutual anonymity* provides both initiator anonymity and responder anonymity. *Unlinkability* means that even if the initiator and responder can each be identified as participating in some communication, they cannot be identified as communicating with each other.

One example of an anonymous system is Anonymizer [22] in which all anonymous paths go through the centralized Anoymizer server. While simple, Anonymizer suffers from the single point of failure problem: It fails if the server reveals a user's identity or if an adversary can observe the server's traffic.

To address the problem of single point of failures, some systems such as Anonymous Remailer [1] and Onion Routing [21] propose to achieve anonymity by having anonymous paths route through a small, fixed core set of *mixes* [3]. However, such systems have several limitations. First, if a corrupt entry mix receives traffic from a non-core node, it can identify that node as the origin of the traffic. Further, colluding entry and exit mixes can use timing analysis to disclose both source and destination. Second, traffic analysis attacks are difficult to counter. Cover traffic has been proposed to deal with such attacks, but it could incur a big bandwidth overhead. Third, the drastic imbalance between the relatively small number of mixes and the potential large number of users might pose a capacity problem. Lastly, legal attacks are another major threat, i.e., law enforcement could prevent institutions from operating a mix.

To overcome the aforementioned drawbacks, systems such as Crowds [12], MorphMix [13] and Tarzan [7] provide anonymity by having messages route through anonymous paths involving a randomly chosen sequence of nodes. In such systems, each node is a mix and an anonymous path can follow any possible path through the system. However, the resulting anonymous paths are vulnerable to node failures: If a node on a tunnel is down, the request/reply message is not able to route through the tunnel to the destination. Consequently, node failures pose a *functionality* problem for anonymous paths.

In this paper, we propose a novel tunnelling approach to achieving anonymity in distributed hash tables (DHTs) [11, 15, 20, 24], called *SurePath*. The basic idea is to decouple anonymous tunnels from "fixed" peer nodes. An anonymous tunnel is composed of a sequence of *relay sets*, each of which has a unique identifier $rsetId$ and is dynamically mapped into a peer node called *relay set agent*. Leveraging DHT routing infrastructure and data

replication mechanism, SurePath makes anonymous tunnels resilient to node failures.

The main motivation of SurePath is to strike a balance between functionality and anonymity in DHTs. Current tunnelling techniques [7, 12, 13] have a problem in maintaining long-standing remote login sessions if a node on a tunnel fails. Moreover, when constructing an anonymous path, they cannot make sure if the anonymous tunnel contains only nodes that remain active with high probability. However, SurePath can support long-standing remote login sessions in the face of node failures. Another application is anonymous email systems. Current tunnelling techniques may fail to route the reply email back to the sender due to node failures along the tunnel, while SurePath can route the reply back to the sender thanks to its robustness in the face of node failures (as will be shown in Section 4.6 by using a reply tunnel). We have implemented SurePath in Java on FreePastry 1.3 [8]. We also provide analysis of anonymity and performance evaluation for SurePath in this paper.

The rest of the paper is structured as follows. Section 2 provides related work and necessary background. Section 3 describes goals and threat model in SurePath. We discuss design of SurePath in Section 4. Section 5 gives security analysis. We present experimental results in Section 6 and conclude the paper in Section 7.

2 Related Work and Background

2.1 Centralized Anonymous Systems

Anonymizer [22] is a centralized anonymous system which provides fast, anonymous, interactive communication services. In this system, all anonymous paths go through a proxy called Anonymizer Server. Such a system fails if the proxy reveals a user's identity or if an adversary can observe the proxy's traffic.

2.2 Mixes-based Anonymous Systems

Many Systems such as Anonymous Remailer [1] and Onion Routing [21] achieve anonymity by having anonymous paths route through a small, fixed core set of *mixes* [3]. Each mix decrypts messages, delays, and reorder messages before relaying them to the next mix.

Onion Routing [21] provides anonymous routing using a dedicated set of "onion routers" that are similar to real-time Chaum Mixes. To send a message in an Onion Routing session, the sender chooses a path of onion routers, then encrypts the message in a layered manner using the public keys of each onion router from the last member of the path, creating an onion. As a message routes through an anonymous path, each onion router removes or adds a layer of encryption, depending upon the direction of traversal of the message. Tor [6], the second generation of Onion Routing, achieves initiator anonymity and responder anonymity by using rendezvous points.

Ogata et al. [10] proposed two schemes that are based on the hardness of factorization and the difficulty of the discrete log problem respectively, to offer robust anonymous tunnels when less than a half of mixes are faulty. SurePath differs from [10] in that it relies on the DHT's routing infrastructure and data replication mechanism to provide resilient anonymous routing.

2.3 P2P-based Anonymous Systems

Many anonymous systems where every node is a mix have been proposed. Crowds [12] aims at providing web-browsing anonymity using random forwarding. The initiator sends the message to a randomly-chosen node called *jondo*. Upon the message, each jondo randomly decides to either send the message to the responder or to forward it to another jondo. Tarzan [7] provides a P2P anonymizing network layer by employing cover traffic. It achieves anonymity with layer encryption and multihop routing similar to Onion Routing. MorphMix [13] uses a collusion detection mechanism to detect colluding mixes. Xiao et al. [23] proposed two protocols for mutual anonymity in hybrid P2P networks.

P^5 [18] uses broadcast channels to achieve mutual anonymity. Nodes join one or more broadcast groups to retain anonymity. P^5 allows users to trade-off the degree of anonymity for communication efficiency. Hordes [19] provides initiator anonymity using multicast. An initiator sends a request to a responder using Crowds or Onion Routing, while the responder multicasts the response to the multicast group that is formed by all the initiators. APFS [17] uses an intermediate proxy and Onion Routing to provide mutual anonymity. We reported our preliminary results of resilient anonymous routing in TAP [25].

Freenet [4] uses probabilistic routing to achieve anonymity. FreeHaven [5] uses both cryptography and routing to provide anonymity.

2.4 Background: DHT Infrastructure

Without loss of generality, we take Pastry/PAST [15, 16] as the example. Other DHTs [11, 20, 24] have the similar characteristics to the ones discussed below.

Pastry is a P2P routing substrate that is efficient, scalable, fault-resistant and self-organizing. Each node in the overlay network has a unique $nodeId$ and a pair of public and private keys. Given a file with a $fileId$, Pastry maps the file into a destination node whose $nodeId$ is numerically closest to the $fileId$. Given an overlay network consisting of N nodes, Pastry can route to the numerically closest node for a given $fileId$ in $O(\log N)$ hops. PAST is a large scale, P2P persistent storage utility layer on top of of Pastry. It employs a replication mechanism to store a file on the k nodes whose $nodeId$s are numerically closest to the file's $fileId$. The k nodes are called *replica set* for the file with $fileId$, and k is called *replication factor*. The k replicas for a file is maintained to increase availability

under node churn. In other words, a file can be located unless all k nodes have failed simultaneously.

3 Design Goals and Threat Model

3.1 Goals

SurePath uses an Internet-wide pool of nodes, numbered in thousands, to relay each other's traffic to gain anonymity. In particular, the goals of SurePath are to meet the following requirements:

1) **Initiator Anonymity:** The identity of an initiator is hidden to all other node including the responder.

2) **Unlinkability:** Identities of the communicating parties (initiators and responders) are hidden to adversaries.

3) **Failure-Resilience:** Anonymous paths are fault-tolerant to node failures on the paths.

4) **Low Latency:** Anonymous communication latency is low. Anonymity should not severely compromise performance.

5) **Responder Anonymity:** SurePath can easily be extended to support responder anonymity, by using an additional level of indirection.

3.2 Threat Model

We assume adversaries control a fraction of nodes in the SurePath network. These compromised nodes collude and share each other's information, attempting to break anonymity of legitimate users by getting control of the anonymous tunnels. The adversaries can observe some fraction of network traffic. There is zero latency for messages sent between colluding nodes.

4 SurePath Design

SurePath uses layered encryption and multi-hop routing: Each hop of an anonymous path removes or adds a layer of encryption depending on the traversal direction of messages. The basic idea behind SurePath is to decouple anonymous paths from "fixed" nodes. Unlike current tunnelling techniques, SurePath defines an anonymous path by a sequence of *relay sets*, each of which is specified by a relay set identifier $rsetId$ instead of an IP address. $rsetId$ is similar to the file identifier $fileId$ in DHTs such as Pastry. Given a $rsetId$, the relay set consists of k nodes whose $nodeId$s are numerically closest to the $rsetId$ (k is the replication factor). The one with $nodeId$ numerically closest to $rsetId$ is called *relay set agent* and the other $k-1$ nodes in the relay set are candidates.

An anonymous path consists of a sequence of relay sets. The relay set agent of a relay set is responsible for decrypting the forwarding path information for a message and forwarding the message to the next relay set. If the relay set agent has failed, the other nodes in the relay set will undertake its responsibilities. Unless all the k nodes in a relay set have failed simultaneously, a relay set is capable of relaying messages successfully. All members in a relay set have a replica of *relay set anchor* (RSA) (Section 4.1), by which the candidate nodes in the relay set can take the place of the relay set agent in case that the agent has failed. Put another way, the RSA is replicated on k different nodes covered by the relay set. Leveraging DHT routing infrastructure and data replication mechanism, SurePath makes anonymous paths resilient to node failures.

In SurePath, a node seeking initiator anonymity generates a small number of RSAs (Section 4.2), deploys the RSAs into the DHT overlay (Section 4.3), forms an anonymous path using a subset of the deployed RSAs (Section 4.5), and sends messages through the resulting anonymous path (Section 4.6).

Figure 1 depicts an anonymous path from the initiator I via relay sets rs_1, rs_2 and rs_3. We denote by $\{X\}_K$ encryption of the content X with a key K. When I sends a message M (which may be encrypted for privacy, e.g., by D's public key) to the destination server D through the anonymous path, it encrypts the message in a layered manner from the last hop of the path with the symmetric keys, which results in $\{rs_2, \{rs_3, \{D, M\}_{K_3}\}_{K_2}\}_{K_1}$. Then, I sends the encrypted message to A_1, which is the relay set agent for rs_1. Upon the message, A_1 removes one layer of encryption using K_1, determines the next relay set according to the identifier in the header, and sends it to A_2, which is relay set agent for rs_2. This process repeats until the tail relay set agent A_3 of rs_3 is reached, which relays the message M to D. As will be discussed later, the corresponding reply is sent back to I using a different anonymous tunnel (called *reply tunnel* which is included in message M by I).

Consider the case when A_1 receives the message from I and is going to send the message to A_2, which has already failed. Relying on P2P routing infrastructure and data replication, A_1 is able to route the message to A_2', which has become the relay set agent for rs_2 after A_2 fails. A_2' then removes one layer of encryption using the symmetric key K_2 from its replica of RSA $< rs_2, K_2, h(PW_2) >$ and sends the message to A_3, allowing the message to continue on the anonymous tunnel.

Having anonymous tunnels consist of an open-ended set of peer nodes, however, introduces a new challenge. An adversary can easily operate several malicious nodes in the system and try to break anonymity of legitimate users by getting full control of their anonymous tunnels. With the replication of RSAs, the probability for colluding nodes to compromise other users' anonymity becomes higher. The main motivation behind SurePath is to strike a balance between functionality and anonymity, and our goal is not to provide perfect anonymity in P2P systems.

SurePath does not employ cover traffic due to several reasons. First, cover traffic is very expensive in terms of

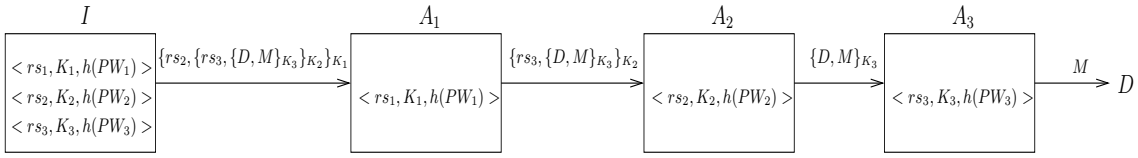


Figure 1: Tunnelling mechanism. rs_i represents the $rsetId$ for the ith relay set. A_i represents the corresponding relay set agent. $< rs_i, K_i, h(PW_i) >$ is the i-th relay set anchor. K_i is a symmetric key.

bandwidth overhead and it does not protect from internal attackers (malicious nodes who act as mixes in SurePath). Second, the number of mixes in SurePath is large (numbered in thousands) and they are probably spread across several countries and ISPs, rendering the global eavesdropper very unlikely. Lastly, the dynamism of P2P systems makes cover traffic hard to maintain.

In what follows, we describe relay set anchors (RSAs), and discuss how to generate and deploy RSAs. Then, we use examples to demonstrate uses of SurePath to achieve initiator anonymity and responder anonymity. We also present a technique to improve anonymous routing performance. Finally, we give a brief discussion on secure routing. Without loss of generality, we take Pastry as the DHT example. However, we believe that our tunnelling approach can be easily adapted to other DHT systems [11, 20, 24]

4.1 Relay Set Anchor

A relay set is "anchored" in the system through a relay set anchor (RSA). A RSA is in the form of $< rsetId, K, h(PW) >$, where $rsetId$ uniquely identifies a relay set and functions as a DHT key for the RSA's storage and retrieval, K is a symmetric key for encryption/decryption, and $h(PW)$ is the hash of a password PW. It can be envisioned a small file stored on the system, where $rsetId$ is the $fileId$, and $\{K, h(PW)\}$ is the file content.

Like a normal file, a RSA is stored on k nodes whose $nodeId$s are numerically closest to its associated $rsetId$. These k nodes are the *replica set* for the RSA and k is the *replication factor*. One of the nodes in the replica set is the relay set agent and the other $k-1$ nodes are the relay set agent candidates. Once the agent fails, one of the candidates will take its place, thus making an anonymous tunnel fault-tolerant to node failures.

The security of RSAs is critical to anonymous tunnels in SurePath. The nodes who have a right to access a RSA must be *restricted*. Only its owner (the initiator who has deployed it) and the nodes in its replica set have the right to access it, while all other nodes cannot be allowed to access it. Any node who wants to access a RSA must be verified that it is either the owner or one of the nodes in the replica set. The identity of an owner can be verified by presenting the corresponding PW of a RSA as will be shown later, while the identity of the nodes in the replica set can be verified due to the verifiable constraint that these nodes' $nodeId$s must be numerically closest to the $rsetId$ of the RSA. It is worth pointing out that a malicious node can disclose the RSAs stored in its local storage to other colluding nodes such that the malicious nodes can pool their RSAs to break anonymity of other users. Section 6 will show evaluation results.

4.2 Generating RSAs

Any node seeking anonymity has to generate and deploy a number of RSAs before using anonymous tunnels. In order to avoid collision in generating RSAs, we propose a RSA generating mechanism which allows a node to generate node-specific RSAs without revealing the node's identity. Note that the uniqueness of a RSA is determined by its $rsetId$. So the $rsetId$ for a given node can be computed from a node-specific identifier $node\_ID$ (which could be, for example, the node's IP address, private key or public key), a secret bit-string $hkey$, and a time t at which the $rsetId$ is created. The purpose of the $hkey$ and t is to prevent other nodes from linking the $rsetId$ with a particular node by performing recomputation of the $rsetId$ upon each node in the system, and revealing that node's identity. The following equation presents the generation more formally:

$$rsetId \leftarrow h(node\_ID, hkey, t).$$

Where h is a uniform collision-resistant hash function, i.e., SHA-1. After generating the $rsetId$ for a RSA, the node then generates a random bit-string as the symmetric key K and another random bit-string as PW.

With the RSA generating approach described above, we can see that, the RSAs a node generates not only avoid collision with those of other nodes, but also prevent other nodes from linking them with the node.

4.3 Deploying RSAs

Before forming a tunnel, a node seeking anonymity must deploy a number of RSAs into the system as the anchors of its relay sets. More importantly, the node must deploy them *anonymously* such that nobody can link a RSA with itself. So, the node needs a bootstrapping anonymous tunnel to deploy the RSAs for its *first* anonymous tunnel. Relying on a public key infrastructure (PKI) on a P2P system by assuming each node has a pair of private and public keys, the node can use Onion Routing

as the bootstrapping tunnel by choosing a set of nodes [1], to deploy the RSAs for its first anonymous tunnel. It creates an onion carrying instructions for each node on the Onion path to store a RSA on the system. For example, a node I creates an onion for the path P_0, P_1, P_2 is $\{P_1, RSA_0, \{P_2, RSA_1, \{D, RSA_2\}_{K_2}\}_{K_1}\}_{K_0}$, where K_i is P_i's public key. It then sends the onion to P_0. Each node on the path removes one layer of encryption and stores the corresponding RSA on the system. Or a node can deploy only one RSA during each Onion Routing session.

It is worth pointing out that Onion Routing is only used to bootstrap a node's first anonymous tunnel. Once the node is able to form the first tunnel using the deployed RSAs, it will use this tunnel to deploy other RSAs if necessary. Without doubt, if a node on the bootstrapping Onion path fails, the deploying process will be aborted. We argue that this is not a problem because the deploying process is not performance critical. A node can always try to use another Onion path to deploy its initial RSAs until the first anonymous tunnel is able to be formed. A node can also rent a trusted node's anonymous tunnels to deploy its initial RSAs. We leave this approach to our future work.

Note that malicious nodes can simply try to flood the system with random RSAs so that "real" RSAs cannot be inserted. This sort of data flooding is a form of denial of service attacks, as it prevents other nodes from deploying RSAs to form anonymous tunnels and gaining anonymity. The usual way of counteracting this type of attack is to charge the node for deploying a RSA. This charge can take the form of anonymous e-cash or a CPU-based payment system that forces the node to solve some puzzles before deploying a RSA.

4.4 Deleting RSAs

Our system provides a mechanism for a node to delete the RSAs which it previously deployed, but no node can delete other nodes' deployed RSAs by using this mechanism. Recall that when a node deploys a RSA, a PW is generated and the $h(PW)$ is included in the RSA. The reason that this value is stored as opposed to just the PW is that it prevents a malicious node from learning the password PW and deleting the RSA. To delete a RSA, a node has to present the secret PW as a proof of the owner. The nodes which store the RSA will hash the received PW, compare the hash value with the stored $H(PW)$, and if they match, remove the RSA from their local storage.

4.5 Forming Tunnels

When forming a tunnel, a node selects a set of RSAs it has already deployed. The chosen RSAs must scatter in the DHT identifier space as far as possible (i.e., with different $rsetId$'s prefixes) to avoid the case that a single node has the information of multiple or all relay sets of the tunnel to be formed.

[1] We can employ the peer selection technique proposed in Tarzan by considering the chosen nodes' IP address prefixes.

4.6 Example: Anonymous File Retrieval

In this section we demonstrate how to use SurePath for an initiator I to anonymously retrieve a file (with fid as its $fileId$) in DHTs such as Pastry.

In the forward path, an initiator I creates a forward tunnel T_f and performs a layered encryption for each relay set. More precisely, consider a forward tunnel T_f that consists of a sequence of 3 relay sets (rs_1, rs_2, rs_3), where rs_i's RSA is $< rid_i, K_i, h(PW_i) >$. Then I produces the message

$$M = \{rid_1, \{rid_2, \{rid_3, \{fid, P_I, T_r\}_{K_3}\}_{K_2}\}_{K_1}\},$$

where P_I is a temporary public key for I and T_r is a reply tunnel for the requested file to route back. T_r is a different tunnel from T_f, consisting of a sequence of 3 relay sets (rs_1', rs_2', rs_3'), where rs_i''s RSA is $< rid_i', K_i', h(PW_i') >$. So

$$T_r = \{rid_1', \{rid_2', \{rid_3', \{bid, fakeOnion\}_{K_3'}\}_{K_2'}\}_{K_1'}\},$$

where $fakeOnion$ is introduced to confuse the last hop in T_r. bid is an identifier subject to a condition that I is the node whose $nodeId$ is numerically closest to it. Therefore, it guarantees that the reply will be route back to I.

To retrieve the file, the initiator I sends the message M to the first relay set agent corresponding to rid_1. The first relay agent node retrieves the symmetric key K_1 from its local storage, removes one layer of encryption using K_1, reveals the next relay set, and sends the extracted message to the next relay set agent. This process continues until the message reaches the tail relay set agent of rid_3. The tail relay set agent strips off the innermost layer of encryption, revealing I's request for file specified by fid. Then it sends the request together with the reply tunnel T_r and P_I' to the responder node R who stores the file f corresponding to fid. Note that P_I and T_r can be encrypted with R's public key for privacy, i.e., $\{P_I, T_r\}_{P_R}$.

Upon receiving the message, the responder R retrieves the file f from its local storage, encrypts f with a symmetric key K_f (i.e., $\{f\}_{K_f}$), encrypts K_f with P_I (i.e., $\{K_f\}_{P_I}$), and sends the $\{f\}_{K_f}$, $\{K_f\}_{P_I}$ and the reply tunnel T_r to the relay set agent of rid_1'. On the reply path, each successive relay set agent removes one layer of encryption from the reply tunnel T_r, revealing the next relay set, and sends $\{f\}_{K_f}$, $\{K_f\}_{P_I}$ and the stripped reply tunnel to the next relay set. This process repeats until the reply message reaches I, which decrypts K_f using the corresponding temporary private key P_I^-, and then decrypts the file f using K_f. Note that each tunnel hop performs only a single symmetric key operation per message.

It is worth pointing out that a request tunnel is different from a reply tunnel in SurePath. This makes it harder for an adversary to correlate a request with a reply.

4.7 Extending to Support Responder Anonymity

In this section, we show how the responder R uses a reply tunnel T_R to achieve responder anonymity in the above example.

In order to serve its file f with identifier fid anonymously, R first hooks fid with a reply tunnel T_R (which is constructed in the same way as T_r) and anonymously stores $< fid, T_R >$ into the node D which is responsible for fid. When the message M from I arrives at the relay set agent of rid_3, it forwards the request together with the reply tunnel T_r and P_I included in M to D. Then, D consults its locally stored $< fid, T_R >$ and routes fid, T_r and P_I through the reply tunnel T_R to R. Upon receiving the message, R sends the file f to I using I's reply tunnel T_r as described above.

Recent work [9] introduced a notion of *extended destination routing* (EDR) which relies on *routing header* to achieve responder anonymity. The reply tunnel used to gain responder anonymity in SurePath essentially serves the same purpose of a routing header, but with enhanced resilience to node failures by decoupling the tunnel from fixed nodes.

4.8 Tunnel Performance Enhancement

Note that routing through an anonymous tunnel of l relay sets involves $l{\cdot}O(\log N)$ overlay hops, introducing a big performance overhead. In this section, we propose a performance enhancement scheme for SurePath's basic tunnelling mechanism.

More precisely, consider a tunnel $T = (rs_1, rs_2, rs_3)$, where rs_i's RSA is $< rid_i, K_i, h(PW_i) >$. For each relay set rs_i, the initiator gets the IP address ip_i of the corresponding relay set agent [2]. Then it creates an encrypted message in the form of

$$\{rid_1, ip_1, \{rid_2, ip_2, \{rid_3, ip_3, \{D, M\}_{K_3}\}_{K_2}\}_{K_1}\}.$$

by embedding the IP address of each relay set agent.

The initiator first attempts to send the message directly to the node with the IP address ip_1. If this node does not exist or it is not the relay set agent of rid_1 any more, it falls back to the DHT routing infrastructure and routes to the relay set agent of rid_1. Each successive relay set agent on the tunnel removes a layer of encryption, revealing the next relay set with a IP address and $rsetId$. It first tries the IP address, if it fails, then routes the message to the relay set agent corresponding to the $rsetId$. This process repeats until the message reaches the tail relay set agent, which in turn routes the message M to the destination node D. Obviously, the tunnelling approach with the IP address embedded as a hint at each hop provides a shortcut to the next relay set agent along the path, resulting in great performance improvement (see Section 6).

[2]The initiator can maintain a cache of the mappings between a tunnel hop rid_i and the IP address of its relay set agent, and it can periodically refresh the cache.

4.9 Discussion: Secure Routing

As discussed earlier, the ability of SurePath in making anonymous tunnels resilient to node failures relies on the DHT routing infrastructure and data replication mechanism. A big concern is how a message can be *securely* routed to a relay set agent given a $rsetId$ in DHT overlays where a fraction of nodes are malicious to pose a threat. Fortunately, we can address the secure routing problem by following the techniques used in [2] — that is, assigning certified $nodeId$s to nodes, maintaining secure routing table and routing messages. The certified $nodeId$s not only prevent nodes from forging $nodeId$s, but also are able to prevent an attacker from easily obtaining a large number of $nodeId$ certificates by requiring some form of real-world currency or solving crypto puzzles. Therefore, the cost of controlling a significant portion of nodes in a large overlay can be made high enough to deter most attackers. Maintaining secure routing tables and using the secure routing tables to forward messages are two parts in routing. To enable secure routing table maintenance, we could impose strong constraints on the set of $nodeId$s that can fill routing table entries. To forward a message, we could apply *routing failure test* to detect problems and then use diverse routes. More detail of secure routing can refer to [2]. In summary, secure routing takes a message and a destination key (e.g.,, $rsetId$) and ensures that with very high probability the message reaches the destination (e.g., the relay set agent) for the key in DHT overlays under the faulty network models.

5 Security Analysis

In this section, we analyze how SurePath can defend against attacks from the various parties in the network. In particular, we focus the analysis on initiator anonymity.

A global eavesdropper: As discussed earlier, SurePath does not employ cover traffic. So if a global eavesdropper can observe every single node in the system, it should be able to break the anonymity of all participants by means of timing attacks at the nodes along anonymous tunnels or end-to-end timing attacks at the first and tail nodes. However, we argue that such an attacker is not realistic in a P2P network with thousands of nodes distributed in the Internet. First, in SurePath each node is a mix and therefore the number of mixes is very large and they are spread across several countries and ISPs. Recent studies [13, 14] also argue that a global attacker is very unlikely in such a P2P system. Second, the dynamism of P2P networks due to node joins and leaves makes it virtually impossible for anyone to get knowledge of the whole network at any time.

A local eavesdropper: An adversary can monitor all local traffic to and from an initiator. Although the eavesdropper will reveal the initiator's traffic patterns (both

sent and received), it cannot figure out the initiator's destination or message content without the cooperation from other nodes.

The responder: The probability that the responder correctly guesses the initiator's identity is $\frac{1}{N-1}$ (N is the number of nodes in system), since all other nodes have the same likelihood of being the initiator.

A malicious node: The mix homogeneity (each node is a potential mix) of our design prevents an adversary from deterministically concluding the identity of an initiator: All nodes both originate and forward traffic. Thus, a malicious node along the tunnel cannot know for sure whether it is the first hop in the tunnel. It can only guess that its immediate predecessor is the initiator with some confidence.

Colluding malicious nodes: We consider the case that an adversary operates a portion of nodes which collude with each other to compromise the anonymity of legitimate users. It can read messages addressed to nodes under its control; it can analyze the contents of these messages. The adversary can use timing analysis to determine whether messages seen at different hops belong to the same tunnel. In SurePath, each relay set anchor RSA is replicated on a replica set of k nodes. If one of these k nodes is malicious, it can disclose the RSA to other colluding nodes. Therefore, malicious nodes can pool their RSAs to break the anonymity of other users. With some probability, the adversary can (1) have the RSAs for all the hops following the initiator along a tunnel (where the first tunnel hop node is under the adversary's control) or (2) control at least the first tunnel hop node and the tail tunnel hop node of a tunnel (in this case, the adversary can use timing analysis attack to compromise the tunnel). Thus, if a message is routed through such corrupt tunnels, the adversary can have a chance to compromise the anonymity. But, it is worth pointing out that the adversary attack on the second case is very limited. This is because, first and most importantly, the adversary does not know if the first hop is really the first hop, which implies he cannot determine who the initiator is. Secondly, the network connection heterogeneity of P2P networks complicates the task of timing analysis attacks. As a result, in Section 6 we mainly focus on the first case.

Note that the primary motivation of SurePath is to strike a balance between functionality and anonymity in very dynamic P2P networks. The adversary may occasionally break the anonymity of a user by using the RSAs he has accumulated, but a user can form another tunnel anyway to protect its future anonymity once its current tunnel is found to be compromised.

6 Experimental Results

We have implemented SurePath in Java on FreePastry 1.3 [8]. FreePastry 1.3 is a modular, open source implementation of the Pastry P2P routing and location substrate. It also includes an implementation of the PAST storage system and the replication manager, which provides application-independent management of replicas by replicating data on the set of k nodes closest to a given key. To be able to perform experiments with large networks of nodes, we implemented SurePath on a network emulation environment, through which the instances of the node software communicate. In all experiments reported in this paper, the peer nodes were configured to run in a single Java VM.

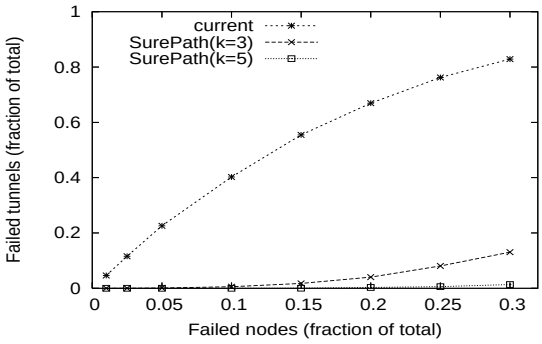


Figure 2: Fraction of tunnels that fail as a function of the fraction of nodes that fail.

6.1 Resilience to Simultaneous Node Failures/Leaves

In the set of experiments, we evaluated the ability of SurePath to function after a fraction of nodes fail/leave simultaneously. We considered a 10^4 node network that forms $5,000$ tunnels, and randomly choose a fraction p of nodes that fail/leave simultaneously. After node failures, we measure the fraction of tunnels that could not function. We define the number of relay sets per tunnel as the *tunnel length*. In this experiment, the tunnel length is 5.

Figure 2 plots the mean tunnel failure rate as a function of p for the current tunnelling techniques, SurePath with the replicator factor $k = 3$, and SurePath with $k = 5$, respectively. Note that in SurePath, there is no significant tunnel failures. A higher replication factor k makes tunnels more robust against node failures. However, in current tunnelling techniques, the tunnel failure rate increases dramatically as the node failure fraction increases.

6.2 Anonymity upon Colluding Malicious Nodes

This set of experiments measured anonymity of SurePath against colluding malicious nodes. The main metric used to evaluate anonymity is the compromised tunnel rate as

a fraction of total tunnels in the system. We considered a 10^4 node network where some of them are malicious and in the same colluding set. We assumed the system with $5,000$ tunnels and randomly chose a fraction p of nodes that are malicious. The tunnel length is 5 by default, unless otherwise specified.

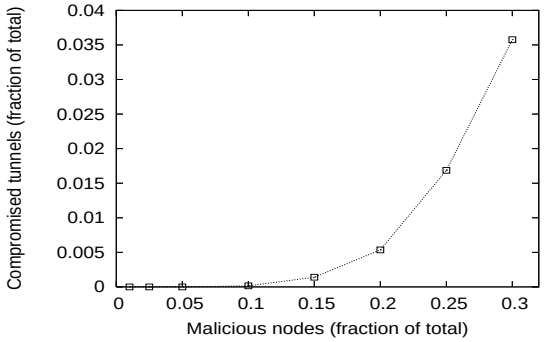


Figure 3: The fraction of tunnels that are compromised as a function of the fraction of nodes that are malicious. The replication factor k is 3.

We first measured the fraction of tunnels that were compromised by malicious nodes. Figure 3 plots the mean compromised tunnel rate as a function of p. As p increases, the corrupt tunnel rate increases. However, there is no significant tunnels compromised even if p is large enough (e.g., 0.3).

In the following experiments, the value of p was fixed to be 0.1. We then evaluated the impact of the replication factor and the tunnel length on anonymity. Figure 4 shows the fraction of tunnels that are compromised as a function of the replication factor. As the replication factor increases, the fraction of tunnels that are compromised increases. This is because a higher replication factor allows malicious nodes to be able to learn more RSAs, increasing the probability of compromising other users' anonymity. Figure 5 shows the fraction of tunnels that are compromised as a function of the tunnel length. Note that the fraction decreases with the increasing tunnel length, and the tunnel length of 5 catches the knee of the curve.

So far our experiments have not considered the dynamism of P2P systems that nodes enter and leave the system at will. Instead of leaving the system, malicious nodes are trying to stay in system as long as possible so that they can accumulate more RSAs to break others' anonymity. For example, if a benign node leaves, its responsible RSAs are taken by another node, which might happen to be a malicious node. Moreover, the DHT's data replication mechanism might happen to make malicious nodes to become the members of some RSAs' replica sets as nodes leave. Therefore, malicious nodes can take advantage of the leaves of other nodes to learn more RSAs. We started a system initially with $5,000$ tunnels. During each time unit, we simulated that a number of 100 benign nodes leaves and then another set of 100 benign nodes

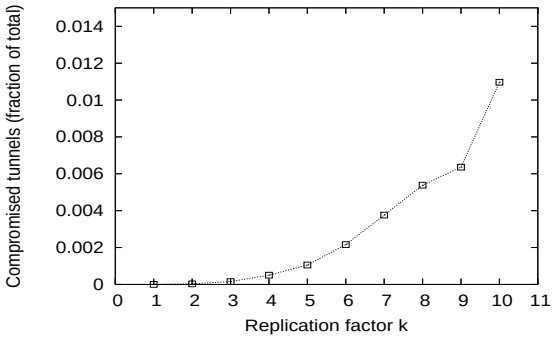


Figure 4: The fraction of tunnels that are compromised as a function of the replication factor.

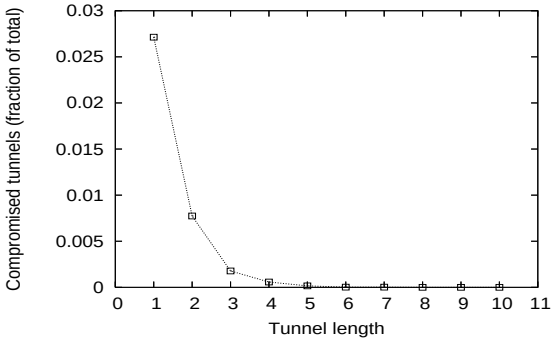


Figure 5: The fraction of tunnels that are compromised as a function of the tunnel length.

join the system, keeping the fraction of malicious nodes p on 0.1 after each time unit. Then, we measured the fraction of tunnels that are compromised after each time unit. Figure 6 plots the mean compromised tunnel rate. "un-refreshed" means that the original $5,000$ tunnels were used throughout the experiment, while "refreshed" means that a new set of $5,000$ tunnels were created to replace the old tunnels after each time unit. Note that the compromised rate of "un-refreshed" increases steadily as time goes, while that of "refreshed" keeps almost constant. We conclude that in such dynamic P2P systems, users must refresh their tunnels periodically to reduce the risk of having their anonymity compromised.

6.3 Performance

In this set of experiments, we evaluated the performance of SurePath in terms of transfer latency between peer nodes. Our performance analysis focused on the overhead introduced by SurePath. We simulated the size of a P2P network from 100 to $10,000$ nodes. Each link in the network had a random latency from 10 ms to 2300 ms, randomly selected in a fashion that approximates an Internet network [17]. All links had a simulated bandwidth of 1.5 Mb/s. A randomly chosen initiator transferred a 2Mb file with a random $fileId$ to a node whose $nodeId$

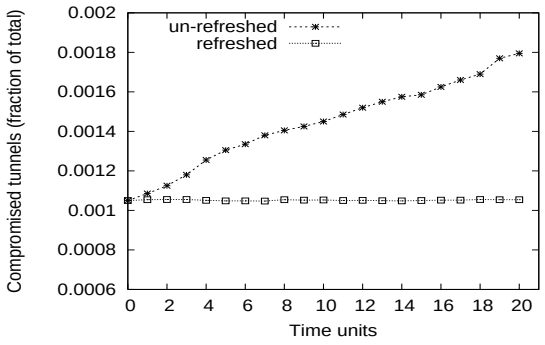


Figure 6: The fraction of tunnels that are compromised. The replication factor k is 5.

is numerically closest to the $fileId$ in the following three ways: (1) *overt* transfer relying on DHT routing infrastructure that does not provide anonymity; (2) anonymous transfer using SurePath; (3) anonymous transfer using performance optimized SurePath, denoted by $SurePath+$ (as discussed in Section 4.8). We ran 30 simulations for each network size, and each of the simulations involved 100, 000 file transfers.

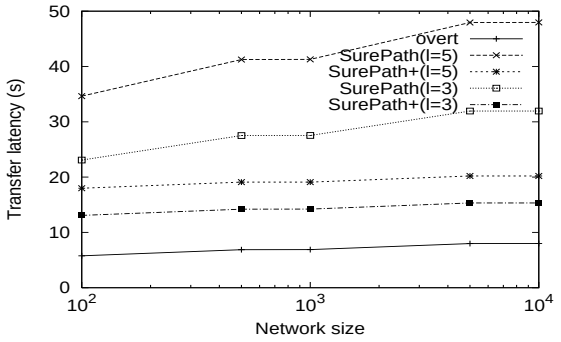


Figure 7: Transfer latencies. l is the tunnel length.

Figure 7 shows transfer latencies as a function of network sizes. Note that SurePath's basic tunnelling mechanism introduces a significant latency penalty in file transfer. A longer tunnel incurs bigger performance overhead, though it provides better anonymity. However, $SurePath+$ can dramatically reduce the latency penalty, improving tunnelling performance. It is worth pointing out that the overhead introduced by symmetric encryption/decryption in tunnelling is negligible in the experiments.

7 Conclusions and Future Work

In this paper, we present SurePath to improve resilience of anonymous routing in dynamic P2P systems. Via detailed simulations, we have arrived at the following conclusions: (1) Leveraging the DHT routing infrastructure and data replication mechanism, SurePath is fault-tolerant to node failures. (2) By carefully choosing the replication factor and tunnel length, SurePath can strike a balance between functionality and anonymity. (3) SurePath's performance optimized tunnelling mechanism can greatly improve routing performance. (4) Users seeking anonymity must reform their tunnels periodically against colluding malicious nodes in dynamic P2P networks to reduce the risk of having their anonymity compromised.

SurePath currently has some limitations. First, unlike MorphMix [13] and Tarzan [7], SurePath lacks the ability to control future hops along a tunnel. It trades this ability for functionality. Second, we have not addressed the admission control problem in SurePath. In securing routing, the certified nodeIds could control the admission of peers, and we believe *trust* management could be used to control the admission and exclude malicious peers from the system. In addition, other incentive mechanisms could possibly be introduced to encourage nodes to protect others' anonymity. Third, SurePath does not have a mechanism to detect compromised tunnels. It requires users to reform their tunnels periodically against colluding malicious nodes. Nevertheless, we believe that SurePath is a first step towards understanding the construction of anonymous tunnels from peers in dynamic P2P systems, and it provides a balance point between functionality and anonymity.

References

[1] Anonymous remailer, http://www.lcs.mit.edu/research/.

[2] M. Castro, A. Ganesh, A. Rowstron, and D. S. Wallach, "Security for structured peer-to-peer overlay networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pp. 299-314, Dec. 2002.

[3] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, pp. 422-426, Feb. 1981.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Workshop on Design Issues in Anonymity and Unobservability*, pp. 331-320, July 2000.

[5] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Workshop on Design Issues in Anonymity and Unobservability*, pp. 67-95, July 2000.

[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, pp. 303-320, Aug. 2004.

[7] M. J. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pp. 121-129, Nov. 2002.

[8] Freepastry, http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/.

[9] M. Kinateder, R. Terdic, and K. Rothermel, "Strong pseudonymous communication for peer-to-peer reputation systems," in *Proceedings of the 2005 ACM symposium on Applied computing*, pp. 1570-1576, 2005.

[10] W. Ogata, K. Kurosawa, K. Sako, and K. Takatani, "Fault tolerant anonymous channel," in *First International Conference on Information and Communications Security*, vol. 1334, pp. 440-444, Springer-Verlag, 1997.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, pp. 161-172, Aug. 2001.

[12] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for Web transactions," *ACM Transactions on Information and System Security*, vol. 1, pp. 66-92, Nov. 1998.

[13] M. Rennhard and B. Plattner, "Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the Workshop on Privacy in the Electronic Society*, pp. 91-102, Nov. 2002.

[14] M. Rennhard and B. Plattner, "Practical anonymity for the masses with mix-networks," in *Proceedings of the twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 255-260, June 2003.

[15] A. Rowstron, and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, pp. 329-350, Nov. 2001.

[16] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pp. 188-201, Oct. 2001.

[17] V. Scarlata, B. N. Levine, and C. Shields, "Responder anonymity and anonymous peer-to-peer file sharing," in *Proceedings of IEEE International Conference on Network Protocols (ICNP'01)*, pp. 272-281, Nov. 2001.

[18] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A protocol for scalable anonymous communication," in *Proceedings of 2002 IEEE Symposium on Security and Privacy*, pp. 58-70, May 2002.

[19] C. Shields, and B. N. Levine, "A protocol for anonymous communication over the internet," in *ACM Conference on Computer and Communications Security*, pp. 33-42, Nov. 2000.

[20] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *Proceedings of ACM SIGCOMM*, pp. 149-160, Aug. 2001.

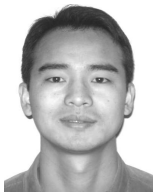
[21] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *IEEE Symposium on Security and Privacy*, pp. 44-54, May 1997.

[22] The anonymizer, http://www.anonymizer.com/.

[23] L. Xiao, Z. Xu, and X. Zhang, "Mutual anonymity protocols for hybrid peer-to-peer systems," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pp. 68-75, May 2003.

[24] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, *Tapestry: An Infrastructure for Fault-Tolerance Wide-area Location and Routing*, Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.

[25] Y. Zhu and Y. Hu, "Tap: A novel tunnelling approach for anonymity in structured P2P systems," in *Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)*, pp. 21-28, Aug. 2004.



Yingwu Zhu received his PhD degree in Computer Science & Engineering from the University of Cincinnati in 2005. He obtained his BS and MS degrees in Computer Science from the Huazhong University of Science and Technology, China, in 1994 and 1997, respectively. He is an assistant professor of Computer Science and Software Engineering at the Seattle University. His research interests include operating systems, file and storage systems, peer-to-peer systems, distributed systems, and sensor networks.

Yiming Hu received his PhD degree in Electrical Engineering from the University of Rhode Island in 1998. He obtained a BE degree in Computer Engineering from the Huazhong University of Science and Technology, China. He is an associate professor of computer science and engineering at the University of Cincinnati. His research interests include computer architecture, storage systems, peer-to-peer systems, operating systems and performance evaluation. He is a recipient of a US National Science Foundation CAREER Award. He is a senior member of IEEE.