TAP: A Novel Tunneling Approach for Anonymity in Structured P2P Systems

Yingwu Zhu
Department of ECECS
University of Cincinnati
zhuy@ececs.uc.edu

Yiming Hu
Department of ECECS
University of Cincinnati
yhu@ececs.uc.edu

Abstract

In this paper we present **TAP***, a novel* **T***unneling approach for* **A***nonymity in structured* **P***2P systems. An important feature of TAP is that anonymous tunnels are fault-tolerant to node failures. Relying on P2P routing infrastructure and replication mechanism, the basic idea behind TAP is to decouple anonymous tunnels from "fixed" P2P nodes and form anonymous tunnels from dynamic tunnel hop nodes. The primary motivation of TAP is to strike a balance between functionality and anonymity in dynamic P2P networks. We have implemented the tunneling mechanism in Java on FreePastry 1.3. An analysis of its anonymity and performance was evaluated via detailed simulations.*

1. Introduction

With the rapid growth and public acceptance of the Internet as a means of communication and information dissemination, *anonymity* is becoming a big concern for many Internet applications, such as anonymous web browsing, anonymous e-mail, and private P2P file sharing. The ultimate goal of anonymization is to protect a participant in a networked application in such a manner that *nobody* can determine his/her identity.

Many anonymous systems (e.g., [1, 18]) have been proposed to protect the identity of the participants. They use a small, fixed core set of *mixes* [3] to form an anonymous path (or tunnel). Such systems, however, 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. Also, colluding entry and exit mixes can use timing analysis to disclose both source and destination. Secondly, traffic analysis attacks are difficult to counter. Cover traffic has been proposed to deal with such attacks, but it hurts performance and introduces a big bandwidth cost. Thirdly, the drastic imbalance between the relatively small number of mixes and the potential large number of users might pose a capacity problem. Lastly, law enforcement could force the operator of a mix to disclose the identities of its users; and an authoritarian regime can block the network access to the small number of mixes.

To overcome the aforementioned drawbacks, peer-to-peer (P2P) based systems such as Crowds [10], MorphMix [11] and Tarzan [7] have been recently introduced to provide anonymity by having messages route through anonymous paths involving a randomly chosen sequence of P2P nodes. In such systems, each node is a mix and an anonymous path can follow any possible path through the system. Unfortunately, such P2P-based environment introduces a new problem: the mixes in a P2P system can join and leave at any time, anonymous paths therefore tend to be less stable. A path fails if one of its mixes leaves the system. Put another way, the dynamic nature of P2P systems renders anonymous tunnels vulnerable to node leaves/failures. If a node on a tunnel is down, the request/reply message is not able to route through the tunnel to the destination/originator. Thus, the dynamism of P2P systems poses a *functionality* problem for anonymous tunnels.

In this paper we present TAP, a novel tunneling approach for anonymity in structured P2P systems [12, 17, 9, 20]. The basic idea is to decouple anonymous tunnels from "fixed" nodes. Leveraging the P2P routing infrastructure and replication mechanism [12, 13], as will be shown later, TAP can make anonymous tunnels fault-tolerant to node failures. For example, current tunneling techniques [10, 11, 7] have a problem in maintaining long-standing remote login sessions, if a node on a tunnel fails. However, TAP can support long-standing remote login sessions in the face of node failures. Another application is anonymous email systems. Current tunneling techniques may fail to route the reply email back to the sender due to node failures along the tunnel, while TAP can route the reply back to the sender thanks to its robustness (as will be shown in Section 4 by using a reply tunnel T_r).

The rest of the paper is structured as follows. Section 2 describes TAP's system architecture and design. Section 3 details TAP's tunneling approach. Section 4 demonstrates how to use TAP to anonymously retrieve a file in structured P2P systems. Section 5 discusses the tunneling perfor-

Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)
0190-3918/04 $20.00 © 2004 **IEEE**

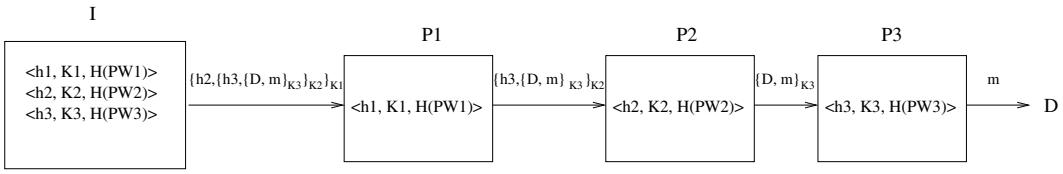

Figure 1. Tunneling mechanism. h_i represents the $hopId$ for the ith hop. P_i represents the tunnel hop node whose $nodeId$ is numerically closest to the hopId h_i. K_i represents the associated symmetric key for the ith hop. $< h_i, K_i, H(PW_i) >$ is the tunnel hop anchor related to the ith hop.

mance enhancement. Section 6 gives TAP's security analysis. Section 7 evaluates TAP's anonymity and performance. Section 8 describes related work, and we finally conclude on Section 9.

2. Architecture and Design

TAP aims to use an Internet-wide pool of nodes, numbered in thousands, to relay each other's traffic to gain anonymity. To access the outside world anonymously, a user sets up an anonymous tunnel, which starts at his own node, via some other nodes. We call the node that is setting up the anonymous tunnel the *initiator*. The last node of the tunnel is called the *tail node*. We also distinguish between *benign nodes*, which are nodes that do not try to break the anonymity of other users and *malicious nodes*, which may collude with each other to compromise others' anonymity.

The basic idea behind TAP is to decouple anonymous tunnels from "fixed" nodes. Unlike current tunneling techniques, TAP defines an anonymous tunnel by a sequence of tunnel hops, each of which is specified by a $hopId$ (hop identifier) instead of a IP address. A $hopId$ is similar to a $fileId$ and represents a peer node whose $nodeId$ is numerically closest to the $hopId$. We call such a node a *tunnel hop node*. Leveraging the P2P routing infrastructure and replication mechanism [1], a tunnel hop node for a given $hopId$ can be located despite the arrival and failure of nodes, unless all k nodes (one node is the tunnel hop node and the other $k-1$ nodes are tunnel hop node candidates for such a $hopId$. k is the replication factor.) have failed simultaneously. Therefore, an anonymous tunnel in TAP is composed of a set of tunnel hops, each of which is mapped into a tunnel hop node *adaptively* as nodes join and leave.

A message routes through an anonymous tunnel using mix-style layered encryption: each hop of the tunnel removes or adds a layer of encryption depending on the direction of traversal of the message. We denote by $\{m\}_K$ encryption of the message m with a symmetric key K. Figure 1 depicts an anonymous tunnel from the initiator I via tunnel hops h_1, h_2, and h_3. When the initiator I sends a message m to the destination server D through the anonymous tunnel, it encrypts the message in a layered manner from the last hop of the tunnel with the symmetric keys, which results in $\{h_2, \{h_3, \{D, m\}_{K_3}\}_{K_2}\}_{K_1}$. Then I sends the encrypted message to P_1, which is the tunnel hop node for h_1. When P_1 receives the message, it removes one layer of encryption using K_1, determines the next hop according to the identifier in the header, and sends it to P_2, which is the tunnel hop node for h_2. This process repeats until the tail node P_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 the *reply tunnel*) which is included in message m by I.

An important feature of TAP is that anonymous tunnels are fault-tolerant to node failures. This is because the k replicas for the tunnel hop anchor $< h_i, K_i, H(PW_i) >$ of each tunnel hop are maintained on k different nodes whose $nodeId$s are numerically closest to h_i in spite of node joins and leaves. For example, consider the case when P_1 receives the message from I and is going to send the message to P_2, which has already failed. Relying on the P2P routing infrastructure and replication mechanism, P_1 is able to route the message to P_2', which has become the tunnel hop node for h_2 after P_2 fails. P_2' then removes one layer of encryption using the symmetric key K_2 and sends the message to P_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 the anonymity of legitimate users by getting full control of their anonymous tunnels. With the replication of the tunnel hop information (i.e., the tunnel hop anchor as will be shown in Section 3.1), the probability for colluding nodes to compromise other users' anonymity becomes higher. However, the main motivation of TAP is to strike a balance between functionality and anonymity, and our goal is not to provide perfect anonymity in P2P systems.

TAP does not employ cover traffic due to the following reasons. First, cover traffic is very expensive in terms of

[1] Due to space constraints, we refer readers to [12, 13] for more details.

bandwidth overhead and it does not protect from internal attackers (malicious nodes who act as mixes in our system). Secondly, the number of potential mixes in our system is large (numbered in thousands or more) and they are probably spread across several countries and ISPs, rendering global eavesdropping very unlikely.

3. Tunneling Approach

In this section we detail TAP's tunneling mechanism. Without loss of generality, we take Pastry/PAST as an example for structured P2P systems. However, we believe that our tunneling approach can be easily adapted to other systems [17, 9, 20, 5, 8].

3.1. Tunnel Hop Anchor (THA)

A tunnel hop is "anchored" in the system through a *tunnel hop anchor*. A tunnel hop anchor is in the form of $< hopId, K, H(PW) >$, where $hopId$ uniquely identifies the tunnel hop and functions as a DHT (distributed hash table) key for the THA'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 $hopId$ is the $fileId$, and $K+H(PW)$ is the file content.

A THA is stored on k nodes whose $nodeId$s are numerically closest to its associated $hopId$. The k nodes are the *replica set* for the THA and k is the *replication factor*. The node with $nodeId$ numerically closest to $hopId$ in the replica set is the tunnel hop node and the other $k-1$ nodes are the tunnel hop node candidates. If the tunnel hop node fails, one of the candidates will take its place, thus making an anonymous tunnel fault-tolerant to node failures.

The security of THAs is critical to anonymous tunnels in TAP. The nodes who have the right to access a THA 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. Any node who wants to access a THA 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 THA 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 $hopeId$ of the THA.

3.2. Generating THAs

Any node seeking anonymity has to generate and deploy a number of THAs before using anonymous tunnels. In order to avoid collision in generating THAs, we propose a THA generating mechanism which allows a node to generate node-specific THAs without revealing the node's identity. Note that the uniqueness of a THA is determined by its $hopId$. So the $hopId$ for a given node can be computed from a node-specific identifier $node\_ID$ (which could be, for example, the node's IP address, its private key or its public key), a secret bit-string $hkey$, and a time t at which the $hopId$ is created. The purpose of the $hkey$ and t is to prevent other nodes from linking the $hopId$ with a particular node by performing recomputation of the $hopId$ upon each node in the system, and revealing that node's identity. The following equation presents the generation more formally: $hopId \leftarrow H(node\_ID, hkey, t)$, where H is a uniform collision-resistant hash function, e.g., SHA-1. After generating the $hopId$ for a THA, the node then generates a random bit-string as the symmetric key K and another random bit-string as the PW.

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

3.3. Deploying THAs

Before forming a tunnel, a node seeking anonymity must deploy a number (e.g., 3-5) of THAs into the system. More importantly, the node must deploy them anonymously such that nobody else can link a THA with itself. Thus, the node needs a bootstrapping anonymous tunnel to deploy the THAs for the *first* anonymous tunnel. Relying on a public key infrastructure on a P2P system by assuming each node has a pair of private and public keys, the node can use Onion Routing [18] as the bootstrapping tunnel by choosing a set of nodes [2], to deploy the THAs for its first anonymous tunnel. It creates an onion carrying instructions for each node on the Onion path to store a THA on the system. For example, a node I creates an onion for the path P_0, P_1, P_2 is $\{P_1, THA_0, \{P_2, THA_1, \{D, THA_2\}_{K_2}\}_{K_1}\}_{K_0}$. It then sends the onion to P_0. Each node on the path removes one layer of encryption and stores the corresponding THA on the system. Or a node can deploy only one THA during each Onion Routing session.

It is worth pointing out that Onion Routing is just used to bootstrap a node's first anonymous tunnel. Once the node is able to form the first tunnel using the deployed THAs, it will use this tunnel to deploy other THAs if necessary. Without doubt, if a node on the bootstrapping Onion path fails, the deploying process will be aborted. We argue 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 THAs 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 THAs. We leave this approach to our future work.

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

Note that malicious nodes can simply try to flood the system with random THAs so that "real" THAs cannot be inserted. This sort of data flooding is a form of denial of service, as it prevents other nodes from deploying THAs to form anonymous tunnels and gaining anonymity. The usual way of counteracting this type of attack is to charge the node for deploying a THA. 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 THA.

3.4. Deleting THAs

Our system provides a mechanism for a node to delete the THAs which it previously deployed. But any node cannot delete other nodes' deployed THAs by using this mechanism. Recall that when a node deploys a THA, a PW is generated and the $H(PW)$ is included in the THA. The reason 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 THA. To delete a THA, a node has to present the secret PW as a proof of ownership. The nodes which store the THA will hash the received PW, compare the hash value with the stored $H(PW)$, and if they match, remove the THA from their local storage.

3.5. Forming Tunnels

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

4. A Sample Application: Anonymous File Retrieval

In this section we demonstrate how to use TAP for an initiator I to retrieve a file (with fid as its $fileId$) in structured P2P systems like Pastry/PAST.

In the forward path, the initiator I creates a forward tunnel T_f, and performs a layered encryption for each tunnel hop. More precisely, consider a forward tunnel T_f that consists of a sequence of 3 hops (h_1, h_2, h_3), where $h_i =< hid_i, K_i, H(PW_i) >$. Then I produces the message $M = \{hid_2, \{hid_3, \{fid, K_I{}', T_r\}_{K_3}\}_{K_2}\}_{K_1}$, where $K_I{}'$ 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. More precisely, it consists of a sequence of 3 hops $(h_1{}', h_2{}', h_3{}')$, where $h_i{}' =< hid_i{}', K_i{}', H(PW_i{}') >$. So $T_r = \{hid_1{}', \{hid_2{}', \{hid_3{}', \{bid, fakeOnion\}_{K_3{}'}\}_{K_2{}'}\}_{K_1{}'}\}$, 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 route back to I.

To retrieve the file, the initiator I sends the message M to the first tunnel hop node corresponding to hid_1. The first tunnel hop node retrieves the symmetric key K_1 from its local storage, removes one layer of encryption using K_1, determines the next tunnel hop node of h_2 and sends the extracted message to the next tunnel hop node. This process continues until the message reaches the tail tunnel hop node of h_3. The tail node strips off the innermost layer of encryption, revealing I's request for file fid. Then it sends the request together with the reply tunnel T_r and $K_I{}'$ to the responder node R who stores such a file f corresponding to fid.

Upon receiving the message, the responder node R retrieves the file f from its local storage, encrypts the file with a symmetric key K_f ($\{f\}_{K_f}$), encrypts K_f with $K_I{}'$ ($\{K_f\}_{K_I{}'}$), and sends the $\{f\}_{K_f}$, $\{K_f\}_{K_I{}'}$ and the reply tunnel T_r with hid_1 removed to the tunnel hop node of $h_1{}'$. On the reply path, each successive tunnel hop node removes one layer of encryption from the reply tunnel T_r, revealing the next tunnel hop, and sends $\{f\}_{K_f}$, $\{K_f\}_{K_I{}'}$ and the stripped reply tunnel to the next tunnel hop node. This process repeats until the reply message reaches I, which decrypts K_f using the corresponding temporary private key ${K_I}^{-1}$, and then decrypts the file f. Note that each tunnel hop performs only a single symmetric key operation per message that is processed.

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

5. Tunnel Performance Enhancement

Note that in a P2P system (e.g., Pastry/PAST) consisting of N nodes, it can route to the numerically closest node for a given $fileId$ in $\lceil \log_{2^b} N \rceil$ hops (b is a system parameter, with a typical value of 4). As a result, the anonymous tunnel of l hops might involve $l * \lceil \log_{2^b} N \rceil$ hops, introducing a big performance overhead.

In this section we propose a performance enhancement scheme for the basic tunneling mechanism. More precisely, consider a tunnel $T = (h_1, h_2, h_3)$, where $h_i =< hid_i, K_i, H(PW_i) >$. For each tunnel hop h_i, the initiator gets the IP address ip_i of the corresponding tunnel hop node [3]. Then it creates an encrypted message in the form of $\{hid_2, ip_2, \{hid_3, ip_3, \{D, M\}_{K_3}\}_{K_2}\}_{K_1}$, by embedding the IP address of each tunnel hop node.

[3] The initiator can maintain a cache of the mappings between a tunnel hop $hopId$ and the IP address of its tunnel hop node, and it can periodically refresh the cache.

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 tunnel hop node of h_1 any more, it routes to the tunnel hop node of h_1 by resorting to the P2P routing infrastructure. Each successive tunnel hop node on the tunnel removes a layer of encryption, revealing the next hop with a IP address and $hopId$. It first tries the IP address, if it fails, then routes the message to the tunnel hop node corresponding to the $hopId$. This process repeats until the message reaches the tail node, which in turn routes the message M to the destination node D. Obviously, the tunneling approach with the IP address embedded as a hint at each hop provides a shortcut to the next tunnel hop along the path, resulting in great performance improvement (as will be shown in Section 7).

6. Security Analysis

In this section we analyze how our tunneling approach can defend against attacks from the various parties in the P2P system. In particular, we focus the analysis on initiator anonymity.

A global eavesdropper: As discussed earlier, TAP does not employ cover traffic due to its expensive bandwidth overhead. 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 analysis at the node along anonymous tunnels or end-to-end timing analysis at the first and tail nodes. However, we argue that such an attacker is not realistic in the P2P environment with thousands of nodes distributed in the Internet. First, in our design each node is a mix and the number of mixes is very large and they are spread across several countries and ISPs, therefore a global attacker is very unlikely in such a P2P environment. Secondly, the dynamism of P2P systems 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 TAP, each THA is replicated on a replica set of k nodes. If one of these k nodes is malicious, it can disclose the THA to other colluding nodes. As such, malicious nodes can pool their THAs to break the anonymity of other users. With some probability, the adversary can (1) have the THAs for all the hops following the initiator along a tunnel, or (2) control at least the first tunnel hop node and the tail tunnel hop node of a tunnel (the adversary can use timing analysis attack to compromise the tunnel). Thus, if a message belonging to these two cases reaches a malicious node, 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 7 we mainly focus on the first case.

Note that the primary motivation of TAP 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 THAs 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.

7. Experimental Results

We have implemented TAP in Java on FreePastry 1.3 [2]. 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 TAP 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.

7.1. Simultaneous Node Failures/Leaves

In this experiment, we evaluate the ability of TAP to function after a fraction of nodes fail/leave simultaneously.

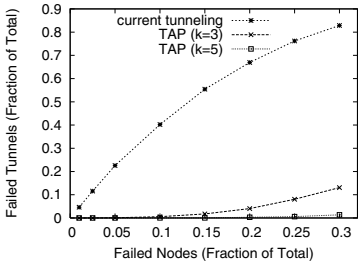

Figure 2. The fraction of tunnels that fail as a function of the fraction of nodes that fail.

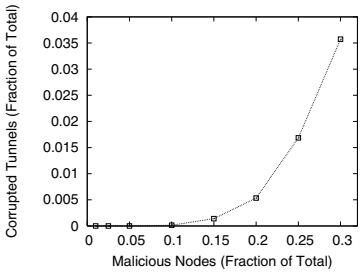

Figure 3. The fraction of tunnels that are corrupted as a function of the fraction of nodes that are malicious. The replication factor k is 3.

We consider a 10^4 node network that forms $5,000$ tunnels, and randomly choose a fraction p of nodes that fail/leave. After node failures/leaves, we measure the fraction of tunnels that could not function. Note that we define the number of tunnel hops 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 tunneling, TAP with the replicator factor $k = 3$, and TAP with $k = 5$, respectively. Note that in TAP, there is no significant tunnel failure. A higher replication factor k makes tunnels more fault-tolerant to node failures. However, in the current tunneling approach, the tunnel failure rate increases dramatically as the node failure fraction increases.

7.2. Colluding Malicious Nodes

We now evaluate the anonymity of TAP against colluding malicious nodes in the system. We again consider a 10^4 node network, where some of them are malicious and in the same colluding set. We assume the system has $5,000$ tunnels and randomly choose a fraction p of nodes that are malicious. The tunnel length is 5 by default, unless otherwise specified.

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

In the following experiments, the value of p is fixed to be 0.1. We then evaluate the impact of the replication factor and the tunnel length on anonymity. Figure 4 (a) shows the fraction of tunnels that are corrupted as a function of the replication factor. As the replication factor increases, the fraction of tunnels that are corrupted increases. This is because a bigger replication factor allows malicious nodes to be able to learn more THAs, increasing the probability of compromising other users' anonymity. Figure 4 (b) shows the fraction of tunnels that are corrupted 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.

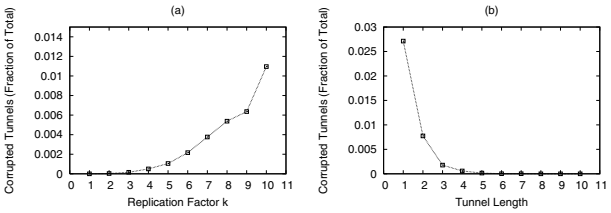

Figure 4. The fraction of tunnels that are corrupted as a function of the replication factor (a) and the tunnel length (b).

So far our experiments have not considered the dynamism of P2P systems that nodes enter and leave the system at will. In the presence of node leaves, malicious nodes instead can try to stay in system as long as possible so that they can accumulate more THAs to break others' anonymity. For example, if a benign node leaves, its responsible THAs are taken by another node, which might happen to be a malicious node. Moreover, the replication mechanism of P2P systems might happen to make malicious nodes to become the members of some THAs' replica sets as nodes leave. As such, malicious nodes can take advantage of the leaves of other nodes to learn more THAs. We assume there are $5,000$ tunnels in the beginning of the system. During each time unit, we simulate that a number of 100 benign nodes leaves and then another set of 100 benign nodes joins the system. So the fraction of malicious nodes p is kept on 0.1 after each time unit. Then we measure the fraction of tunnels that are corrupted after each time unit. Figure 5 plots the mean corrupted tunnel rate. "unrefreshed" means that the original $5,000$ tunnels are used

throughout the experiment, while "refreshed" means that a new set of $5,000$ tunnels are created to replace the old tunnels after each time unit. Note that the corrupted rate of "unrefreshed" increases steadily as time goes, while that of "refreshed" keeps almost constant. We conclude that in such dynamic P2P systems, users should refresh their tunnels periodically to reduce the risk of having their anonymity compromised.

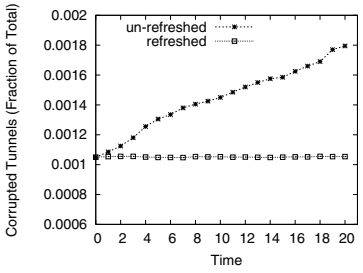

Figure 5. The fraction of tunnels that are corrupted. The replication factor k is 5.

7.3. Performance

In this section we evaluate the performance of TAP in terms of transfer latency between peer nodes. Our performance analysis focuses on the overhead introduced by TAP. 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 [14]. 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 $nodeId$ is numerically closest to the $fileId$ in the following three ways: (1) overt transfer relying on the P2P routing infrastructure ("overt"), (2) anonymous transfer using TAP's basic tunneling mechanism ("TAP_basic"), and (3) anonymous transfer using TAP's performance optimized tunneling mechanism ("TAP_opt") (as discussed in Section 5). We ran 30 simulations for each network size, and each of the simulations involved $100,000$ file transfers.

Figure 6 shows the transfer latency. Note that TAP's basic tunneling mechanism introduces a significant latency penalty in the file transfer. A longer tunnel introduces bigger performance overhead, thought it provides better anonymity. However, TAP's performance optimized tunneling mechanism can dramatically reduce the latency penalty, thus greatly improving the tunneling performance. It is also worth pointing out that the overhead introduced by symmetric encryption/decryption in tunneling is negligible in this experiment.

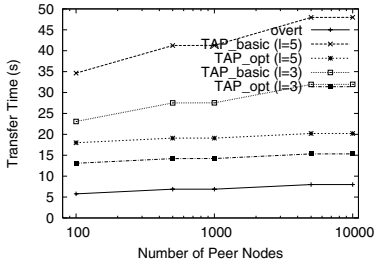

Figure 6. Transfer latency. l is the tunnel length.

8. Related Work

In this section we present the research work related to anonymity and P2P systems.

Many systems such as Anonymous Remailer [1] and Onion Routing [18] achieve anonymity by having anonymous paths route through a small, fixed core set of *mixes* [3]. Recently, anonymous systems where every node is a mix have been proposed, such as Crowds [10], Hordes [16], Tarzan [7], MorphMix [11], and P^5 [15]. Recent work [19, 14] aims at mutual anonymity between an initiator and a responder. However, all these systems have not addressed the functionality problem of anonymous paths.

Structured P2P systems [12, 17, 9, 20], provide a P2P routing and lookup infrastructure. And P2P storage and file systems (e.g., [13]) layer their storage on top of such structured P2P systems. Currently, we implement TAP in Pastry/PAST, but our tunneling approach can be easily adapted to other systems [17, 9, 20, 5, 8]. There are two anonymous storage systems that deserve mentioning. Freenet [4] uses probabilistic routing to achieve anonymity, and FreeHaven [6] uses both cryptography and routing to provide anonymity.

9. Conclusions and Future Work

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

The ability of TAP in making anonymous tunnels fault-tolerant to node failures is to rely on the P2P routing infrastructure and replication mechanism. A big concern is how a message can be *securely* routed to a tunnel hop node given a $hopId$ in P2P overlays where a fraction of nodes are malicious to pose a threat. Due to space constraints, we refer readers to our extended report [21] for the details of secure routing.

TAP currently has its own limitations. First, unlike MorphMix [11] and Tarzan [7], TAP lacks the ability to control future hops along a tunnel. It trades this ability for functionality. Secondly, TAP does not have a mechanism to detect corrupted/malicious tunnels. It requires users to reform their tunnels periodically against colluding malicious nodes. In our next steps, we hope to address these issues. Nevertheless, we believe that TAP is a first step towards understanding the construction of anonymous tunneling from P2P nodes in dynamic systems, and it provides a balance point between functionality and anonymity.

10. Acknowledgments

This work is supported in part by National Science Foundation under Career Award CCR-9984852 and ACI-0232647, and the Ohio Board of Regents.

References

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

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

[3] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):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*, pages 331–320, Berkeley, CA, July 2000.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, Banff, Canada, Oct. 2001.

[6] 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*, pages 67–95, Berkeley, CA, July 2000.

[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 2002)*, pages 193–206, Washington, D.C., November 2002.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201, Cambridge, MA, Nov. 2000.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, San Diego, CA, Aug. 2001.

[10] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, Nov. 1998.

[11] 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*, Washington, DC, November 2002.

[12] 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)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[13] 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)*, pages 188–201, Banff, Canada, Oct. 2001.

[14] 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 2001)*, Riverside, CA, Nov. 2001.

[15] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A protocol for scalable anonymous communication. In *Proceedings of 2002 IEEE Symposium on Security and Privacy*, pages 58–70, Berkeley, CA, May 2002.

[16] C. Shields and B. N. Levine. A protocol for anonymous communication over the internet. In *ACM Conference on Computer and Communications Security*, pages 33–42, Athens, Greece, Nov. 2000.

[17] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, CA, Aug. 2001.

[18] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, Oakland, California, May 1997.

[19] 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*, pages 68–75, Providence, Rhode Island, May 2003.

[20] 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.

[21] Y. Zhu and Y. Hu. TAP: A novel tunneling approach for anonymity in structured P2P systems. Technical report, Department of ECECS, University of Cincinnati, Sept. 2003.