Low-Cost and Reliable Mutual Anonymity Protocols in Peer-to-Peer Networks

Li Xiao, *Member*, *IEEE*, Zhichen Xu, *Member*, *IEEE*, and Xiaodong Zhang, *Senior Member*, *IEEE*

Abstract—We present several protocols to achieve mutual communication anonymity between an information requester and a provider in a P2P information-sharing environment, such that neither the requester nor the provider can identify each other, and no other peers can identify the two communicating parties with certainty. Most existing solutions achieve mutual anonymity in pure P2P systems without any trusted central controls. Compared with two such representative ones, our protocols improve efficiency in two different ways. First, utilizing trusted third parties and aiming at both reliability and low-cost, we propose a group of mutual anonymity protocols. We show that with some limited central support, our protocols can accomplish the goals of anonymity, efficiency, and reliability. Second, we propose a mutual anonymity protocol which relies solely on self-organizations among peers without any trusted central controls. In this protocol, the returning path can be shorter than the requesting path. This protocol does not need to broadcast the requested file back to the requester so that the bandwidth is saved and efficiency is improved. In addition, this protocol does not need special nodes to keep indices of sharing files, thus eliminating the index maintenance overhead and the potential for inconsistency between index records and peer file contents. We have evaluated our techniques in a browser-sharing environment. We show that the average increase in response time caused by our protocols is negligible, and these protocols show advantages over existing protocols in a P2P system.

Index Terms—Peer-to-peer (P2P) systems, mutual anonymity, communication protocols, Internet systems, overlay networks.

1 Introduction

One important problem in peer-to-peer (P2P) systems is to enforce the trust of the data stored in the system and the security of the peers. In a P2P system, each peer can play three different roles: as a *publisher* to produce documents, as a *provider* (or a *responder*) to host and deliver documents upon requests, as a *requester* (or an *initiator*) to request documents. In some systems, a provider and a publisher can be the same peer for the same document. In some other systems, a provider and a publisher are different peers for the same document for various reasons. For example, a publisher can distribute its documents to other provider peers in order to resist censorship, and documents can also be cached in some nonproducer peers.

Depending on the circumstances, applications and users of a system may require different levels of anonymity. It is desirable, in practice, that the identity of a publisher be hidden to resist censorship (publisher anonymity), or that either a responder or an initiator be anonymous (responder or initiator anonymity), or that both responder and initiator be anonymous (mutual anonymity). In the most stringent version, achieving mutual anonymity requires that neither the initiator nor the responder can identify each other, and no other peers can identify the two communicating parties with certainty.

- *L. Xiao is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824. E-mail: lxiao@cse.msu.edu.*
- *Z. Xu is with Hewlett-Packard Laboratories, Hewlett-Packard Company, Palo Alto, CA 94304. E-mail: zhichen@hpl.hp.com.*
- *X. Zhang is with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: zhang@cs.wm.edu.*

Manuscript received 12 Aug. 2002; accepted 15 May 2003. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 118637.

P2P systems can be classified into two classes: pure P2P systems, where peers share data without a centralized coordination, and hybrid P2P, where some operations are intentionally centralized, such as indexing of peers' files. Which form the system takes makes a difference. For instance, in a hybrid P2P, whether the indexing servers can be trusted or not has a critical implication on how anonymity is enforced.

Our goal is to achieve mutual anonymity between the initiator and responder with high efficiency. We consider two cases. In the first case, we assume the existence of trusted index servers (e.g., Napster [12] and browser-aware proxies [29]). In our work, instead of having both the initiator and responder each prepare their own covert path, we rely on the index server to prepare a covert path for both of them, reducing operations and communication overhead. We have proposed two new techniques: *center-directing*, where encryption cost is independent of the length of the covert path, and *label-switching* that eliminates potentially excessive messages in center-directing (Section 3).

In the second case, we assume a pure P2P setting. We propose an anonymity protocol called *shortcut-responding* that can greatly reduce communication overhead while preserving mutual anonymity (Section 4).

We analyze our proposed protocols in Section 5. We present our empirical experience of the techniques in a browser-sharing environment in Section 6. We discuss how to select the protocols based on their merits and limits from different aspects in Section 7. We conclude in Section 8.

2 Related Work and Motivation to Our Work

The related work includes existing protocols for the three types of anonymity. We have paid special attention to the

1045-9219/03/$17.00 © 2003 IEEE Published by the IEEE Computer Society

work on mutual anonymity, which has motivated us to develop new protocols.

2.1 Publisher and Sender Anonymity

Publisher Anonymity: In order to protect a publisher peer, many systems provide a censorship resistance facility. In Freenet [3], each node in the response path may cache the reply locally, which can supply further requests and achieve publisher anonymity. Publius [26] splits the symmetric key used to encrypt and decrypt a document into n shares using Shamir secret sharing and stores the n shares on various peers. Any k of the n peers must be available to reproduce the key. Instead of splitting keys, FreeHaven [4] and [18] split a document into n shares and store them in multiple peers. Any k of the n peers must be available to reproduce the document. Tangler [25] and Dagster [23] make newly published documents depend on previously published documents. A group of files can be published together and named in a host-independent manner.

Initiator/Responder Anonymity: Most existing anonymity techniques are for client/server models, which only hide the identities of the initiator (clients) from the responder (the server), but not vice versa. Anonymizer [9] and Lucent Personalized Web Assistant (LPWA) [8] act as an anonymizing proxy between a user and a server to generate an alias for a user, which does not reveal the true identity of the user. Many systems achieve sender anonymity by having messages go through a number of middle nodes to form a covert path. In Mix [2] and Onion [24], the sender part determines the covert path, and a message is encrypted in a layered manner starting from the last stop of the path. Instead of having the initiator select the path, Crowds [14] forms a covert path in such a way that the next node is randomly selected by its previous node. Hordes [21] applies, that is, a similar technique used in Crowd, but it uses multicast services to anonymously route the reply to the initiator. Freedom [7] and Tarzan [6] are similar to Onion Routing, but they are implemented at IP layer and transport layer rather than the application layer.

2.2 Existing Mutual Anonymity Protocols: Their Merits and Limits

Our study targets on mutual anonymity between an initiator and a responder. There are two most related and recent papers aiming at achieving mutual anonymity: Peer-to-Peer Personal Privacy Protocol (P^5) [20] and Anonymous Peer-to-Peer File Sharing (APFS) [17].

Sherwood et al. [20] first propose to use a global broadcast channel to achieve mutual anonymity, where all participants in the anonymous communication send fixed length packets onto this channel at a fixed rate. Noise packets can be used to maintain a fixed communication rate. Besides enforcing both initiator and responder anonymity, this protocol pays special attention to eliminate the possibility of determining the communication linkability between two specific peer nodes by providing equal and regular broadcast activities among the entire peer group. The broadcast nature of this framework can limit the size of the communication group. To address this limit, the authors further propose the P^5 scheme that creates a hierarchy of broadcast channels to make the system scalable. Different levels of the hierarchy provide different levels of anonymity at the cost of communication bandwidth and reliability. As authors stated in this paper, P^5 will not provide high bandwidth efficiency. But, P^5 allows individual peer to trade off anonymity degree and communication efficiency.

In the APFS system, a coordinator node is set to organize P2P operations. Although this node is not considered as a highly centralized and trusted server, it should be on service all the time, and it plays an important role to coordinate peers for file sharing. APFS allows new peers to join and leave the system periodically by sending a message to the coordinator. Willing peers begin anonymously announcing themselves as servers to the coordinator. After contacting the coordinator, peers anonymously and periodically send lists of files using alias names to those servers. An initiator peer starts to request documents by anonymously querying the coordinator for available servers. The coordinator responds with a list of current servers. A peer then anonymously sends queries to some servers. Upon requests, these servers will send back N matches to the initiator peer. The initiator sends the match request to a path where the tail node is the last member. The tail node then forwards the request to the responder and returns the reply back to the initiator. APFS uses Onion as the base to build their protocol. There are two advantages for APFS. First, all the communications in the system are mutual anonymous. Even the coordinator does not know the physical identities of the peers. Second, the anonymous protocols are designed for a pure P2P, where the trusted centralized servers may not be available.

However, there are also several disadvantages associated with APFS solely relying on volunteering. First, the suitability of a volunteering peer needs to be taken into account, which can significantly affect the performance of P2P systems. To do so, the coordinator needs to examine each volunteering peer before possibly assigning a task, such as peer indexing. The background checking of peers has to be done anonymously, increasing the communication overhead. Second, the number of servers can be dynamically changed. In the worst scenario, no qualified peers are available for a period of time, causing the P2P system to be in a weak condition. Third, since any peer can be a server, a malicious node can easily become a server. Although the peer identities are hidden from a server, a server has the power to provide wrong indexing information to mislead the initiators. Finally, since no trusted servers are available, the anonymous communications have to be highly complicated.

Both P^5 and APFS provide unique solutions to achieve mutual anonymity in pure P2P systems without any trusted central controls. We believe that limited trusted and centralized services in decentralized distributed systems are desirable and necessary. In practice, trusted central parties exist and effectively function, such as proxies and firewalls in Internet and distributed systems. Utilizing these trusted parties and aiming at both reliability and low cost, we propose a group of mutual anonymity protocols. We show that, with some limited central support, our protocols can accomplish the goals of anonymity, efficiency, and reliability. We have also proposed a mutual anonymity protocol solely relying on self-organizations among peers without any trusted central controls. In this protocol, the returning path can be shorter than the requesting path. Comparing with P^5, this protocol does not need to

broadcast the requested file back to the requester so that the bandwidth is saved and efficiency is improved. Comparing with APFS, this protocol does not need special nodes to keep indices of sharing files, thus eliminating the index maintenance overhead and the potential problem of inconsistency between index records and peer file contents.

3 Anonymity with Trusted Third Parties

We present our techniques for achieving mutual anonymity of the initiator and responder with the help of trusted index servers that keeps (but not publicize) the whereabouts of the contents that are stored in the peers. Each peer sends an index of files they are willing to share with others peers to selected index servers periodically, or when the percentage of updated files reaches to a certain threshold. We use I to represent the initiator, R to represent the responder, S to represent the index server that I contacts, and p_i $(i = 1, 2, \ldots)$ to represent a peer. For conciseness of the presentation, we assume there is only one index server. Section 3.4 discusses how multiple index servers will be involved in order to scale a P2P system.

A simple solution is to have an index server act as an anonymizing proxy hiding the identities of I and R from each other and other peers. But, this index server may become a bottleneck making the system not scalable. Instead, we have the index server randomly select several peers to act as a middle node. These middle nodes form a covert path for the peer that possesses the content to send the content to the peer that requests the content.

We describe one intuitive protocol using mix, and two new protocols, *center-directing* and *label-switching*, which are advanced alternatives. In the rest of the paper, we use $X \rightarrow Y : M$ to represent X sending a message M to Y. We use K_X to denote the public key of X and $\{M\}K$ to represent encrypting the message M with the key K.

3.1 A Mix-Based Protocol: An Intuitive Solution

The detail of the mix-based protocol is shown below:

Step 1: The initiator sends a request to S. The request is encrypted with S's public key.

$$I \rightarrow S : \{file\_ID\}K_S.$$

Step 2: S finds out that the file is possessed by R, it selects a list of peers $p_0, p_1, \ldots, p_k$ at random, and builds a mix with R as the first member of the path, I as the last member, and with p_i in the middle. We call this path mix. mix is of the form $(p_0, (p_1 \ldots (I, fakemix)K_{p_k} \ldots)K_{p_0})K_R$. The item $fakemix$ is introduced to confuse the last node in the mix, p_k, so that the format of a message passing through the middle nodes are the same. So, p_k cannot be sure that she is the last stop. In addition, it generates a DES key K. It then sends a message to R. The message includes K encrypted with R's public key, $\{file\_ID\}$ encrypted with the DES key K, K encrypted with I's public key, and the mix.

$$S \rightarrow R : \{K\}K_R, \{file\_ID\}K, \{K\}K_I, mix.$$

Step 3: R obtains K using its private key to decrypt $\{K\}K_R$; it uses K to decrypt the portion of the message $\{file\_ID\}K$ and gets the file f based on the $file\_ID$; it uses its private key to peel mix to obtain p_0, and also the rest of the path, mix', i.e., $(p_1 \ldots (I, fakemix)K_{p_k} \ldots)K_{p_0}$. It encrypts the file f with K and sends a message to p_0:

$$R \rightarrow p_0 : \{f\}K, \{K\}K_I, mix'.$$

Step 4: p_i decrypts mix' using its private key to obtain the address of the next member in the mix paths, and this also produces the rest of the path, mix''. It then sends a message to p_{i+1}. For p_k, p_{k+1} is I.

$$p_i \rightarrow p_{i+1} : \{f\}K, \{K\}K_I, mix''.$$

Step 5: I obtains K using its private key and uses K to decrypt the encrypted file.

We omitted the details on how the initiator knows that the content is destined to it. This must be done efficiently. There are three alternatives: 1) to have S also encrypt $file\_ID$ with the I's public key and have R send this along with the content, 2) to encrypt a magic number and the DES key with I's public key, or 3) to encrypt $file\_ID$ in $fakemix$ using the I's public key. In the remainder of the paper, we assume that our protocols choose one of the above alternatives.

The anonymizing path is selected by the trusted index server, and the mix routers are selected among the peers. Having the index server perform a path selection, this scheme becomes less vulnerable to traffic analysis since the peers' public keys need only be exposed to the index server. Otherwise, an eavesdropper who knows the peers' public keys may reconstruct the path by applying the public keys in a reverse order. Furthermore, the index server has the opportunity to balance the load of the peers that act as mix routers. In this protocol, only the path is encrypted with an expensive public key encryption, and the content is encrypted with a less expensive DES key. This arrangement makes the scheme efficient. This scheme can be made more efficient by encrypting the mix path using secret keys that are shared between the index server and each of the peers. The content is encrypted by a key that is generated by the index server and is only known to I and R. This hides the content from anybody except I and R.

To defend against traffic analysis, S can have the responder pad the contents, and the middle nodes can encrypt the DES-encrypted message pairwise so that a message appears different along the path. These enhancements can be done to all our protocols. Fig. 1 shows an example with two middle nodes.

3.2 Center-Directing

Alternatively, S can be used to reduce the number of encryption/decryption operations. We describe two new protocols: *center-directing* and *label-switching*.

Instead of passing the mix through the whole covert path in the *mix-based* protocol, the *center-directing* protocol has the index server send each node in the covert path its next hop individually. The basic idea of the center-directing protocol is as follows: The index server S selects several peers to form a covert path. It directs the content through the path by sending each middle node p_i a pair $< label(p_i), p_{i+1} >$ that is encrypted with p_i's public key. The labels can be generated such that $label(p_{i+1}) = \{label(p_i)\}K_{p_{j_{i+1}}}$. The labels uniquely identify a message, and p_{i+1} is the next member in the covert path. When the peer p_i sees a message from a

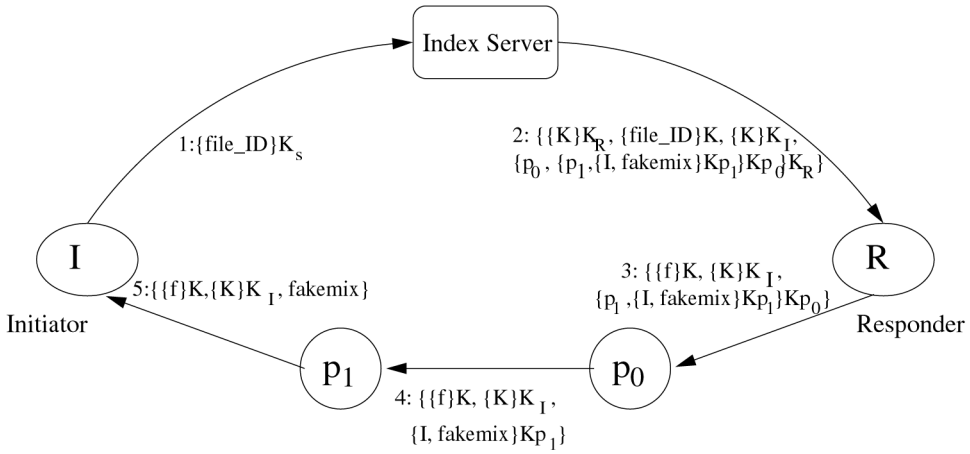

Fig. 1. An example of the mix-based protocol.

peer labeled "l," it will change the label to $\{l\}K_{p_{j_{i+1}}}$ and forward the message to p_{i+1}. Each p_i keeps a hash table to synchronize between the message from the index server and the message from its previous hop. The $p_{j_{i+1}}$ is a random generated node number. Using the random node's public key to encrypt the request label each time, we can defend against traffic analysis in the sense that 1) labels for the same request appear differently along the covert path, and 2) the randomly generated node has no correlation with the nodes in the covert path. This protocol takes advantage of the fact that encryption cost is much lower than decryption cost in public key encryption. In contrast to the mix-based scheme, this protocol uses messages to set up the path. Although this incurs additional cost in hashing, setting up the path can be done in parallel. The big difference lies in the size of items being encrypted and decrypted. The server needs to encrypt $k < label, p_i >$ pairs. Each peer decrypts once to reveal the next hop and encrypts once to produce a label for the next hop. Therefore, the sizes of items that need to be encrypted by public key encryption are independent of the path length.

The details of the protocol are shown below:

Step 1: The initiator I sends a request to S.

$$I \rightarrow S : \{file\_ID\}K_S.$$

Step 2: S first generates k; that is, the number of middle nodes in the covert path. S then generates a unique label for the request, n, and the first middle node in a covert path, p_0. S also generates a DES key K. In addition, it randomly generates another node number used to convert the request label in node R, p_{j0}. S then sends the following message to R:

$$S \rightarrow R : \{K\}K_R, \{n, file\_ID, p_0, p_{j_0}\}K, \{K\}K_I.$$

Step 3: S generates the next stop of p_0, p_1, and another random node number p_{j_1}. It converts the request label n to $\{n\}K_{p_{j_0}}$. S then sends a message to node p_0:

$$S \rightarrow p_0 : \{n\}K_{p_{j_0}}, \{p_1, p_{j_1}\}K_{p_0}.$$

Step 4: R obtains K using its private key to decrypt $\{K\}K_R$; it uses K to decrypt the portion of the message $\{file\_ID\}K$ and gets the file f based on the $file\_ID$; it converts the request label n to $\{n\}K_{p_{j_0}}$. It encrypts the file f with K and sends a message to p_0:

$$R \rightarrow p_0 : \{n\}K_{p_{j_0}}, \{f\}K, \{K\}K_I.$$

Step 5: S generates the next stop of p_i $(i > 0)$, p_{i+1}, and another random node number $p_{j_{i+1}}$. It converts the request label $\{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_{i-1}}}$ to $\{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_i}}$. For p_k, p_{k+1} is I. S then sends a message to node p_i:

$$S \rightarrow p_i : \{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_i}}, \{p_{i+1}, p_{j_{i+1}}\}K_{p_i}.$$

Step 6: p_i first matches the request label coming from the index server and the request label coming from last stop, $\{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_i}}$, so that it finds the next stop for the request, p_{i+1}. It then converts the request label $\{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_i}}$ to $\{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_{i+1}}}$, and sends a message to p_{i+1}. For p_k, p_{k+1} is I.

$$p_i \rightarrow p_{i+1} : \{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_{i+1}}}, \{f\}K, \{K\}K_I.$$

Step 7: I obtains K using its private key and uses K to decrypt the encrypted file.

Fig. 2 illustrates this protocol with two middle nodes. Each middle node uses an encryption operation to compute the label for setting up the path instead of using a decryption operation.

3.3 Label-Switching

The label-switching protocol further reduces the messaging overhead of center-directing by putting more states on the peers. Rather than sending the middle nodes labels and next hop addresses on-the-fly, the index server produces a path table beforehand. The table is produced such that each peer p_i, as a destination, is associated with several path options. The path is of the form $p_x - p_y - \cdots p_i$ (L). This table is broken into subtables and distributed to peers (encrypted with their public keys). The subtable of p_j consists of a list of pairs of the form $(L, nexthop)$. For every appearance of p_j in the path table, $\ldots - p_j - p_w - \ldots$ (L), the pair (L, p_w) is added to p_j's subtable.

Table 1 shows an example path table with four options for each peer. Table 2 shows some subtables derived from Table 1. In this example, each path option has two middle nodes. The number of middle nodes is not fixed in our

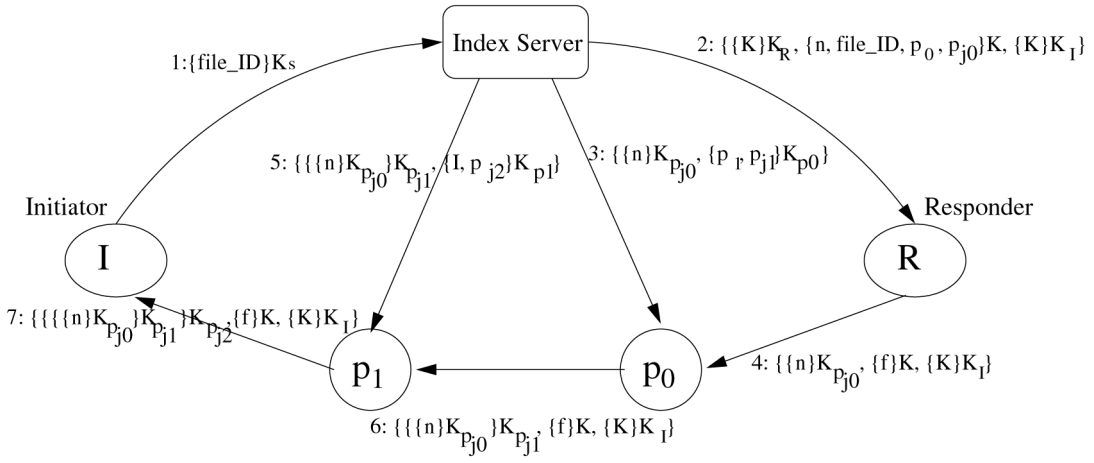

Fig. 2. An example of the center-directing protocol.

design. It has already been shown that variable path-length strategies perform better than fix-length strategies [11]. Assuming that the index server needs to prepare a path from node 5 to 0, it can select among four paths from entry for node 0: *2-3-0(L8)*, *4-6-0(L3)*, *3-4-0(L4)*, and *1-7-0(L1)*. Suppose $L4$ is picked. The message will route to node 3, 4, and, finally, to 0, with each peer using their own subtables.

The detail of the protocol is shown below:

Step 1: The initiator I sends a request to S.

$$I \rightarrow S : \{file\_ID\}K_S.$$

Step 2: S randomly selects a path in the entry for I in the path table (say, $p_0 - p_1 \ldots p_k - I$), and a key K. Assuming that this path has a label l. It sends the following message to R:

$$S \rightarrow R : \{l, p_0\}K, \{K\}K_R, \{K\}K_I.$$

Here, p_0 is the first middle node in the path.

Step 3: R sends a message (the label) to p_0:

$$R \rightarrow p_0 : l.$$

A persistent connection will be established between R and p_0 if the connection does not already exist. This connection is bound to the label l. Each p_i sends a message to p_{i+1} that is obtained from the subtable of p_i.

$$p_i \rightarrow p_{i+1} : l.$$

A persistent connection is set between p_i and p_{i+1}.

Step 4: A message is sent along the persistent connections from R to I.

$$R - l \rightarrow I : \{f\}K, \{K\}K_I.$$

We use $-l \rightarrow$ to represent the persistent connection identified by the label l.

Step 5: I obtains K using its private key and uses K to decrypt the encrypted file.

This protocol does not need the synchronization associated with center-directing protocol; it does not need as much encryption/decryption operations compared with the mix-based protocol: The only encryption and decryption occurs during the subtable distribution. The overhead comes from the spaces for storing the path table and subtables and the time spending on table look-ups. Even though the path table kept in the trusted index may be a target of attack, multiple paths for a given source-destination pair adds one additional level of defense.

To simplify our presentation, we have assumed that we use a single label for the entire path. This protocol can be improved for stronger anonymity by introducing a pair of labels (like the center-directing protocol) for each hop rather than using a single label for the entire path, so that a label appears differently along the covert path. One concern about this protocol is that the static path table may decrease the anonymity degree of label-switching protocol. This concern can be addressed by updating the path table periodically during a light network load period so that the efficiency of label-switching will not be compromised.

3.4 Multiple Trusted Index Servers

In order to scale a P2P system to a large size, we will use multiple trusted index servers. Since multiple proxy servers are always available in an Internet region, this arrangement can be easily set up in practice. Besides scalability, the arrangement of multiple index servers will improve the reliability of a P2P system. As a peer node joins a P2P system,

TABLE 1
Path Table

Peers	path			
0	2-3-0(L8)	4-6-0-(L3)	**3-4-0(L4)**	1-7-0(L1)
1	…	…	…	…
2	…	…	…	…

TABLE 2
Subtables

Peer1		Peer2		Peer3		Peer4	
L1	7	L8	3	L8	0	L3	6
…	…	…	…	**L4**	**4**	**L4**	**0**
…	…	…	…	…	…	…	…

it will register itself in multiple index servers. Servers may be down sometimes, but unlikely at the same time. Thus, the indexing service is fault tolerant and much more reliable than the system with a single index server. However, use of multiple index servers also raises a load balancing issue. Without proper scheduling and redirections of peer requests, the workloads among the index servers can be unbalanced, generating some hot spot servers and leaving some others idle or lightly loaded.

We will adapt our own load sharing schemes [28] to make resource allocations in the P2P system. Each index server node maintains a current load index of its own and/or a global load index file that contains load status information of other index server nodes. The load status can be the number of registered peers, the average number of handled requests, storage for index of files to be shared, and so on. There are two alternatives to balance the workloads among the indexing servers **when a peer wants to join the system**.

- *Index-server-based selections.* When a peer node joins the system and asks for an indexing service, it first randomly selects an index server. The load sharing system may periodically collect and distribute the load information among all index server nodes. Based on the load information of all index server nodes, the selected server will then suggest a list of lightly loaded index servers, including or excluding itself, for the peer node to be registered. One advantage of this approach is reliability. When a peer node leaves the system, it will inform one of the index nodes. This node will carry this message when it broadcasts its load status to other index server nodes. Since all index servers are trusted, a selection of most lightly loaded servers is guaranteed. One disadvantage of this approach is that the global load statuses have to be updated frequently among all the index servers to keep each node informed.
- *Peer-node-based selections.* When a peer node joins the system and asks for an indexing service, it first broadcasts its request to all the index servers. Each index server will then return its status back to the peer node. The peer node will select a list of index servers to be the hosts, which are hopefully the most lightly loaded. When a peer node leaves the system, it will broadcast this status change message to all the index server nodes. In contrast to the alternative of index-server-based selections, this alternative does not require updating the load statuses globally among the index servers because a peer node will collect them each time it needs them. However, reliability is not guaranteed because peer nodes are not trusted, and they may not follow the load balancing principle when they select index server nodes.

There are also two alternatives **when a peer node requests a file**. The first alternative is straight forward. The peer node simply sends the request to index servers one by one. When it reaches the index server that has the index of the requested file, the file will be anonymously delivered to the peer node from a path arranged by the index server. The second approach involves two steps. The peer node first broadcasts a query message to all the index servers. The index servers that have the indices of the requested file will inform the peer node about their service availability. The peer node will then send the request to the index server that has responded earliest, for an anonymous file delivery. If the index server does not deliver the file for some reason, the peer node will try to send the request to other index servers that responded later than the first one. Although broadcast is not involved in the first alternative, the search is not as objective as the second alternative. In general, we have no strong reasons favoring one approach over another. A detailed study of alternatives of multiple index servers is beyond the scope of this paper.

4 Anonymity in Pure P2P

We now describe a technique to achieve mutual anonymity in a pure P2P setting without any trusted third party. We call it *shortcut-responding protocol*. In this protocol, a peer along the requesting path can elect itself to receive document on behalf of the initiator, thereby shortening the returning path.

We describe the details below:

Step 1: The initiator I randomly selects a list of peers, $r_0, r_1, \ldots, r_{kr}$ and builds a one-time $replyblock$ with I as the last member of the path and with r_i in the middle. The remailer $replyblock$ is of the form $(r_{kr}, (r_{kr-1} \ldots (r_0, (I, fakemix)K_{r_0})K_{r_1} \ldots)K_{r_{kr}})$. The I also generates a one-time pair of private key and public key (K_I) associating with the request. Then, I randomly selects a peer, p_0, sends it the message:

$$I \rightarrow p_0 : \{r, replyblock, K_I\},$$

where r encodes the request.

Step 2: A peer p_i can elect itself to act as a $relay$ of the returning path with a probability pv. We call pv the *shortcut probability*. If p_i has not elected itself, the request remains as $\{r, replyblock, K_I\}$. If p_i has self-elected, the $replyblock$ and the request will be left in this node and the request format is changed to $\{r, relay : p_i, K_I\}$. It then decides whether to select p_{i+1} or broadcast the request with probability pb. If p_i has decided to broadcast the message, it will mark the message to avoid broadcasting it multiple times. Therefore, for p_i, the requests it can receive is one of the two formats: $format1 : \{r, replyblock, K_I\}$ or $format2 : \{r, relay : p_i, K_I\}$.

Step 3: If p_i cannot find the content in local storage, it will save the request. We call p_i as R if p_i finds the content in local storage. R encrypts the found file content using K_I.

If R has the request format of $format1$, R contacts the first node in the $replyblock$, r_{kr}, then sends the encrypted file through the $replyblock$ to I.

$$R \xrightarrow[replyblock]{} I : r, \{f\}K_I.$$

If R has the request with the $format2$, it selects a list of peers $o_0, o_1, \ldots, o_{ko}$ at random and builds an $Onion$ with o_0 as the first member of the path, $relay$ as the last member, and with o_i in the middle. The $Onion$ is of the form $(o_0, (o_1 \ldots (o_{ko}, (relay, fakemix)K_{o_{ko}})K_{o_{ko-1}} \ldots)K_{o_0})$. The R first sends the encrypted file through the $Onion$ to the $relay$. If the request has not been discarded in the $relay$, the $relay$ then sends the encrypted file through the $replyblock$ to

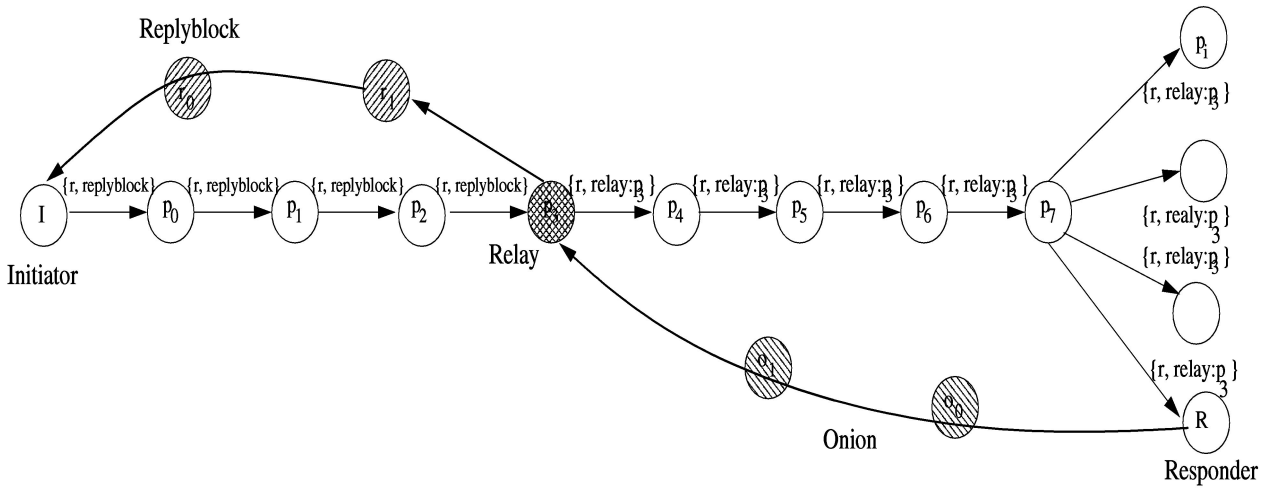

Fig. 3. An example of the shortcut-responding protocol.

I. It discards the request so that duplicated responses can be dropped.

$$R \xrightarrow[Onion]{} relay \xrightarrow[replyblock]{} I : r, \{f\}K_I.$$

Step 4: I uses her private key to decrypt the encrypted file.

Fig. 3 illustrates the protocol with an example. Peer p_3 elects itself as a $relay$ to receive the content on behalf of I. The peer that possesses the content R sends the response to p_3 through the $Onion$. The peer p_3 further sends the response to I through the $replyblock$.

This protocol has several advantages:

1. The response path can be shorter than the requesting path because a peer who receives the request and has the content will send the content through an $Onion$ and a $replyblock$, instead of going through the requesting path to the initiator.
2. Duplicated responses can be discarded earlier.
3. The protocol does not need special nodes to keep indices of sharing files like APFS, thus eliminating the index maintenance overhead, and the potential problem of inconsistency between index records and peer file contents.
4. The protocol does not need to broadcast the requested file like P^5 while it still keeps mutual anonymity, so the efficiency is improved compared with P^5.
5. The protocol uses $replyblock$ that is also used in FreeHaven [4], where the responder contacts directly to the $replyblock$ so that the first stop in the $replyblock$ knows who the responder is. In contrast, *shortcut-responding* protocol has the responder send the requested file to a $relay$ through an $Onion$, and then has the $relay$ send the file back to the initiator through the $replyblock$ so that nobody in the requesting path and responding path can guess the identity of initiator and responder with certainty.

The initiator and responder also cannot guess each other with certainty. Here is a related question to ask. If a node with a request of $format1$ finds the requested file, it then contacts the $replyblock$ and sends back the file. In this case, can the first stop in the $replyblock$ guess the one who contacts her is the responder? The answer is no because the first stop in the $replyblock$ cannot distinguish whether the one who contacts her is the responder or a $relay$. Here is another proposed alternative. Upon receiving a request with $format1$, if a peer node realizes that the requested file is locally allocated, she will not send the file through the $replyblock$ because the first stop in the $replyblock$ can guess that the one who has just been connected is the responder. Instead of immediately providing the file, this peer forwards the request again. But, this particular request is marked by her so that she will accepts a later broadcast request. As soon as she receives this request again from a broadcast, she sends the file back through the $Onoin$ and $replyblock$.

5 Analysis

We analyze the degree of anonymity each protocol can achieve and compare their costs in terms of numbers of encryption/decryption operations.

5.1 Security Analysis

We analyze how the different protocols can defend against attacks from the various parties in the P2P networks. Because the situations for the initiator and the responder are symmetric, we consider only the initiator's *anonymity degrees* that is defined as probabilities that different parties guess the identity of the initiator.

The responder: To the responder, all other peers have the same likelihood of being the initiator. The probability that the responder correctly guess the identity of the initiator is $\frac{1}{n-1}$ (n is the total number of peers). Instead of making a random guess, the responder can bet that the peer to whom she sends the message is the initiator. She is only able to make the right bet if there is no middle node selected. We assume that the probability of existing k middle nodes is $p(k)$, and the probability for the responder to make the right bet is $p(0)$.

A middle node: We consider two cases: In the first case, the middle node makes a random guess because the only

TABLE 3
Comparison of Protocols with k Middle Nodes in Each Covert Path

Protocols		Mix-based	Center-directing	Label-switching	Shortcut-responding
DES	path	$1,1$	$1,1$	$1,1$	$0,0$
(Encrypt, Decrypt)	content	$1,1$	$1,1$	$1,1$	$0,0$
RSA	path	$4+k, 4+k$	$4+3k, 3+k$	$3,3$	$2k$
(Encrypt, Decrypt)	content	$0,0$	$0,0$	$0,0$	$1,1$

thing she is sure about is that she is not the initiator. In this case, the probability to make a correct guess is $\frac{1}{n-1}$. In the second case, the middle node bets that the peer to which it sends the message is the initiator. If there are k middle nodes, only one of the k middle nodes will make a correct bet. The probability for a middle node to make the correct bet is $\frac{1}{n-2}\sum_{k=1}^{n-2}\frac{p(k)}{k}$, and $p(k)$ is the probability of existing k middle nodes.

In both cases, the probability will become smaller if multiple peers communicate simultaneously. For the protocols with the index server, even if a middle node can figure out who is communicating with whom, it still cannot figure out the content of the communication.

A local eavesdropper: An eavesdropper is an entity that can monitor all local traffic. The worst case is when there is only one pair communicating (or the messages being communicated are so distinctive such that the eavesdropper is able to figure out who is communicating with whom). Even in this worst case, the eavesdropper still cannot figure out the content without the cooperation either from the responder or initiator (for the protocols with the index server) or one of the middle nodes (for the shortcut-responding protocol).

Cooperating Peers: We consider cases where at least two middle nodes cooperate, and the responder (or the initiator) can be a collaborator. Two things make it hard for cooperating nodes to guess the identity of the initiator: 1) the middlemen do not know for sure how many communications are proceeding simultaneously, and 2) the format of a message passing through the middle nodes is the same. If k collaborating peers were to make a random guess, the probability for them to make the right guess is $\frac{1}{n-k}$ because all but the k peers can be the initiator. If the collaborating peers were to make a bet, there are two extreme cases. One extreme case is that all the k peers are continuously connected right after the initiator. The probability for them to make a right bet in this case is $\frac{1}{d_0}$, where d_0 is the number of links to the initiator, which have incoming messages simultaneously. The other extreme case is that all noncollaborating peers are distributed among the collaborating peers and the last peer is a collaborating peer. The probability in this case is $\frac{1}{d_0}\prod_{p\in C}\frac{1}{d_i}$, where C is the set of noncollaborating peers, d_i is the number of links to the noncollaborating peer i, which have incoming messages simultaneously. Thus, the probability for k collaborating peers to correctly bet the identity of the initiator is inbetween the above probabilities in two extreme cases.

For all protocols, we can add the following operations to increase the anonymity degree by introducing more confusions. The protocols prepare multiple covert paths for each request. The responder splits the requested file in multiple parts. The parts of the file can be sent back to the initiator through different covert paths. The different parts of the file can be easily combined, based on sequence numbers given by the responder. The Shamir algorithm [19] can also be borrowed to split and combine files, with which a file can be split into n parts and any k parts of them can reproduce the original file.

5.2 Cost of the Different Protocols

In Table 3, we summarize and compare the costs of the protocols in terms of numbers of encryption/decryption operations.

For the center-directing protocol, the time spent on RSA for setting up the anonymizing paths can be less than that of mix-based protocol for two reasons. First, RSA encryption is much faster than RSA decryption. Center-directing uses more encryption than decryption operations. Second, some steps are parallelizable. For the example in Fig. 2, Steps 3 and 4, and Steps 5 and 6. The messages transferred in Steps 3 and 5 are smaller than those in Steps 4 and 6, so Steps 3 and 5 may be finished before Steps 4 and 6.

6 Performance Evaluation

We estimate the additional overhead incurred in the protocols for achieving mutual communication anonymity. Our testbed is the browser-sharing environment where clients share cached Web contents [31]. The clients are the peers, and the proxy server is the index server. The proxy maintains an index of all files that are possibly cached in its clients' browser caches. If a user request misses both in the client's local cache and in the proxy cache, the proxy will search the index file in an attempt to find the file in another client's browser cache. If the file is found in a client's cache, the proxy can then instruct this browser to forward the file to the requesting client. Our metric is the additional response time for each request hit in a remote browser cache compared with the response time of a request hit in the local browser cache. The increment comes from two major sources: time spent on transferring the requested data from the remote cache to the local cache and time spent on the protocols.[1]

We use trace-driven simulations and the Boeing traces [1] for the evaluation. We selected two days' traces (March 4 and

1. We have neglected the costs for building and looking up the hash tables because the hashing cost is insignificant comparing with the other costs.

TABLE 4
Latency

Traces	Total Workload Service Time	# Files Transferred among peers	File Size of total files in Column 3	Data Transfer Time via 2 middle nodes (% of Column 2)	Data Transfer Contention Time for Bus (% of Column 5)
Boeing, 3/4	86,398.9 s	12,647	612 MB	177.82 s (.21%)	0.00003 s (.00002%)
Boeing, 3/5	86,175.8 s	9,868	607 MB	149.64 s (.17%)	0.005 s (.0034%)

March 5, 1999). There are 3,996 and 3,659 clients involved in these two days' traces, representing the total numbers of requests of 219,951 and 184,476, respectively. The total requested file sizes for the two traces are 7.54 and 7.00 Gbytes.

The results show that the average increment of the response time caused by the protocols is trivial. We present detailed performance results in the sections that follow.

6.1 Data Transfer Time through Peer Nodes

We estimate the data transfer time through peer nodes based on a 100 Mbps Ethernet in our simulation. The bus contention is handled as follows: If multiple clients request bus service simultaneously, the bus will transfer documents one by one in FIFO order distinguished by each request's arrival time. Our experiments based on the ping facility show that the startup time of data communications among the clients in our local area network is less than 0.01 second. Setting 0.01 second as the network connection time, Table 4 presents the intranetwork data transfer time for each trace. We can see that the amounts of data transfer times and the bus contention times spent for communications among clients on both traces are very low.

6.2 Overhead of DES and RSA

The source programs of DES and RSA are obtained from [15]. The machine we used for the experiments is a PC with a 1,000 MHz Pentium III CPU and 128 Mbytes of memory. We used a large number of cached files in Microsoft's IE5 browsers as the input files for the tests. We ran each test 10 times. The average of 10 measurements is used.

The running times of DES are proportional to the sizes of the input files. Our measurement results show that DES's speed is 43.3 Mbps. The ratio of the RSA's running time to the input file size is not linear. RSA can encrypt/decrypt at a speed of 543/45.4 Kbps with a 512-bit value, 384/24.8 Kbps with a 768-bit value, and 275/14.6 Kbps with a 1024-bit value. It should be noted that the decryption speed of RSA is 12-19 times slower than the encryption speed. These measured results and the results in Tables 3 and 4 are used in our simulations to calculate the overheads of DES and RSA.

6.3 Additional Storage

The label-switching protocol requires additional storage to keep the path table in the index server and subtables in the peers. We allocate two bytes for each peer identification and two bytes for each path identification. The two bytes can represent up to 65,536 different identifications. For each entry of destination described in Fig. 1, 26 bytes are required in the index server. For the trace with 3,996 peers, the total storage for the path table is 26*3996, which equals to 101 Kbytes. There are a total of 3996*4 paths, and four bytes are needed for each entry of a path in a subtable (see Fig. 2). The storage needed for each peer is less than 3996*4*4, which equals to 62Kbytes. These storage requirements are sufficiently small for the path table and subtables to be held in memory for quick accesses.

6.4 Comparisons of Protocols

We have shown the data transfer times and the costs of DES and RSA operations. Here, we compare the accumulated overheads of the protocols. Fig. 4 compares the total increased response times and their breakdowns for the protocols using the "Boeing March 4 trace" and "Boeing March 5 trace," with two and five clients acting as middle nodes.

The performance results in Fig. 4 show that center-directing and label-switching protocols generate very low overhead, while the other two have relatively higher overhead. The label-switching protocol shows its best performance. It is not desirable if the response time of a request hit in a remote browser cache is larger than that of the same request to the server. This is not a concern because our experiments show that the average response time increment is less than 8.4 ms when we use five middle nodes for both traces. The two protocols with lower overhead only increase the response time to about 2.7 ms when five middle nodes are used.

The time spent on RSA for the mix-based protocol increases as the number of middle nodes increases. In contrast, the times spent on DES and RSA for the center-directing and label-switching protocols are independent of the number of middle nodes. The number of RSA operations of center-directing protocol is the highest (see Table 3). However, most of them are low-cost encryption operations for small messages (such as a request, labels, node IDs) which are parallelizable. Both center-directing and label-switching protocols show very good scalability.

Compared with the other three protocols, the time spent on RSA is considerably high for the shortcut-responding protocol. This is because we use a public key to encrypt the response content that is usually much larger than a message like a request, a label, or a path. The efficiency can be improved if we encrypt the response content with DES keys that are encrypted using public keys in a pair-wise fashion. The traffic analysis can be defended, but the content will be exposed to all middle nodes in a covert path.[2] The RSA cost is high, but it is a constant. So, the shortcut-responding protocol scales well.

2. In all of our protocols, the response content is only visible to the initiator and responder, but is not to any other nodes.

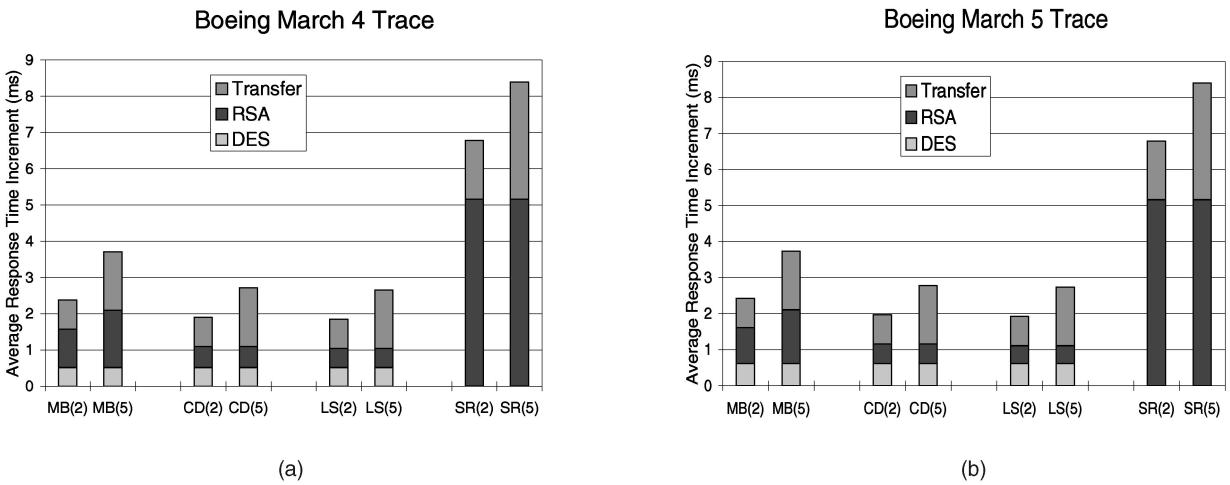

Fig. 4. Breakdown of data transfer and protocol overhead with two and five middle nodes for Boeing March 4 trace (a) and Boeing March 5 Trace (b). $MB(k)$ represent mix-based protocol with k middle nodes. Similarly, CD, LS, and SR represent center-directing, label-switching, and shortcut-responding, respectively.

The data transfer time increases proportionally to the increase of the number of middle nodes. The transfer time of label-switching is lower than that of other protocols because it uses a persistent channel for continuous data transfers between the the same pairs of sending and receiving nodes. The data transfer time is still a dominant portion of the total overhead. We should limit the number of middle nodes to balance the two basic goals: achieving mutual anonymity and quick response time. Guan et al. [11] show that the anonymity degree may not always monotonically increase as the length of communication path increases.

7 Discussions of the Proposed Protocols

We have analyzed a mix-based scheme and several new protocols along with our empirical experience. We now discuss how to select the protocols based on their merits and limits under different conditions.

How to select protocols by considering both efficiency and anonymity degree. The shortcut-responding protocol is designed for pure P2P systems, while mix-based, center-directing, and label-switching protocols are designed for hybrid P2P systems to achieve mutual anonymity. For a pure P2P system, the shortcut-responding protocol can be a good candidate, and its cost can be controlled by properly selecting the number of middle nodes in covert paths.

For a system with a trusted third party, such as a proxy and a firewall, this party can be utilized to provide some centralized support. With such a limited support, both reliability and efficiency of mutual anonymity protocols can be significantly improved.

If storage space is not a concern, the label-switching protocol is the best choice in terms of efficiency. In fact, the storage requirement of this protocol can be acceptable for systems of moderate size (thousands of nodes). The other advantages of this protocol are: It uses a very small number of encryption/decryption operations, and it does not need to keep all private keys in the third party, which can be vulnerable if the third party is attacked. Although the third party keeps a path table, there are multiple options for each destination, and the path table can be updated periodically. Therefore, even if the path table is exposed, it can still be very hard for an attacker to figure out which path is used for a specific data transfer.

If storage space is limited, the center-directing protocol is a good candidate. The mix-based protocol can be used if the RSA costs are tolerable.

Unlike the mix-based protocol, the cipher costs of center-directing and label-switching protocols are independent of the path length. In the case that a large number of middle nodes are required to enforce strong anonymity, center-directing and label-switching are the best choices.

What if a node in a covert path is down? All covert-path based protocols can have this problem. The center-directing protocol could handle this case very well. Since the trusted index server dynamically generates the next node in a covert path, it is easy for the index server to generate another node when it finds that the node it just generated is down.

APFS, shortcut-responding, and mix-based protocols share the same concern for this problem. APFS and the shortcut-responding protocol use Onion as the base. A selected Onion passes through a whole covert path.

In the mix-based protocol, the trusted index server generates a mix that also needs to pass through the whole covert path. When a node in the covert path is down, the communication path needs to be recovered. One solution for this is to let the initiator send the request again when it cannot get response within a certain period of time. Another covert path will be selected, in which all middle nodes are alive, hopefully.

In the shortcut-responding protocol, if the $relay$ cannot get response within a certain period of time, it will send back a message of "NO RESPONSE." When the initiator receives a message of "NO RESPONSE," it means that the $Onion$ part is down and the $replyblock$ part works. If the initiator cannot get anything within a certain period of time, she cannot judge which path is down (maybe both are down). The request has to be sent again. Because the $replyblock$ and $Onion$ are one-time paths, hopefully all the selected nodes for the new request to form these paths are alive.

In APFS, for some initial requests, such as a request to volunteer to be a server, a request to ask for servers, or a request to update index, the requests will be resent if they cannot get response within a certain period of time. For an initiator who already gets N matches for its request, there are also two covert paths between the initiator and responder. One is a path between the responder to its tail node and another one is a path between the tail node and initiator. Because the communications are two-directional, the initiator cannot judge which path is down if she cannot get response, even with the help of the tail node. So, the initiator has to send the same match request or another match request again. But, the initiator will not need to start from the very beginning to request volunteer servers. APFS is more advanced than the shortcut-responding protocol in the sense that it will not sacrifice too much efficiency when a node in a covert path is down. But, APFS still cannot compete with the center-directing protocol because only one covert path needs to be handled in center-directing protocol.

Label-switching protocol generates a path-table in a trusted index server beforehand, and peers keep relevant portions of the path table as subtables. Although the path table and subtables are updated periodically for security reasons, the protocol has to trade off efficiency if a middle node is down. One solution for this is to let the initiator send the request again with a note to the trusted index server that its first request for the same file has not been responded to. When the initiator cannot get a response within a certain period of time, the index server will select a different covert path in the path table. Hopefully, all middle nodes in this covert path are alive.

What if a file cannot be found as it is supposed to be? All index-based protocols, such as mix-based, center-directing, label-switching, and APFS, can have this problem. The index servers keep an index of files that peers are willing to share. The indices are updated by the peers periodically. It is possible that the file has already been replaced in a peer, but the index still shows its existence.

When this happens in the mix-based, center-directing, and label-switching protocols, the responder simply informs the trusted index server that she cannot find the file. The index server then will contact another peer who has the file or send back a message of "NO FILE FOUND" to the initiator. Another alternative is that the responder sends the message of "NO FILE FOUND" to the initiator through the covert path as usual. Then, the initiator sends the request again to the index server with a note that the responder cannot find the file.

In APFS, the responder replies a message of "NO FILE FOUND" to initiator. Since the initiator was responded N matches for her request, she will just try to get the file from another match if she cannot get the file from the first match.

Comprehensively considering all factors, the *center-directing* protocol is the best with some limited support of central control. If efficiency has a high priority over reliability, the *label-switching* and *shortcut-responding* protocols work well for a system with a trusted third party, and a system without any central controls, respectively.

8 Conclusion

Providing a reliable and efficient anonymity protection among peers is highly desirable in order to build a scalable and secured P2P systems. In this paper, we have presented several protocols to achieve mutual anonymity in a P2P file-sharing environment. Our first group protocols take advantage of the existence of trusted third parties to improve efficiency and reliability, and use them to prepare the covert paths for anonymous communications. The other proposed protocol, shortcut-responding, combines both broadcast and self-organizing covert path techniques to achieve mutual anonymity in pure P2P systems without any trusted central controls. After several hop-to-hop requests, this protocol broadcasts the request that is normally a small message. It then sends back the requested file back to the initiator through a dynamically created covert path instead of broadcasting, achieving both communication anonymity and efficiency.

The protocols utilizing trusted third parties may have three potential limits. First, these trusted third parties may become single points of failure. This potential problem can be addressed by our proposed methods of multiple index servers. In addition, we can enforce anonymous communications between any peer to the trusted servers, hiding their identities and locations.

Second, one may have a concern about scalability of P2P system with the involvement of trusted parties. Specifically, we may not have enough trusted parties to handle the increasingly growing Internet user community. We believe this is not a necessary concern. The client/server model will continue to play its important roles and continue to coexist with the P2P model. Thus, the number of trusted servers will proportionally increase as the number of peers increases.

Finally, A P2P system with the involvement of trusted parties may not be completely open and free, but may put some restrictions to peers. For example, a peer has the freedom to join and leave a pure P2P system anytime. Although a peer still has this freedom in our system, she needs to do registration to a predefined index server(s). In fact, we view the involvement of the trusted parties for this respect positively. Researchers in the distributed system community have made a long-term effort to attempt to build trustworthy systems out of untrusted peers. We believe that this principle applies to P2P systems.

The performance and robustness of a P2P system to a great extent depend on the capacity of trusted servers and the suitability of peers to act as middle nodes. A strong P2P system should be self-organizing and adaptive to dynamic application demands and changing of network conditions. When a peer is used for some centralized function (e.g., index servers), some reputation system must be used to regulate their use. We attempt to follow these principles in designing our protocols.

We are looking into combining different approaches to further and synergistically achieve the goal for both strong anonymity and high communication efficiency, as well as to adapt to application needs and network conditions.

ACKNOWLEDGMENTS

This work was partially completed while Li Xiao was a research intern at HP Labs in the Summer of 2001. This work is supported in part by the US National Science Foundation under grants CCR-9812187, EIA-9977030, and CCR-0098055, and by a USENIX Research Scholarship. The work is a part of an independent research project sponsored by the US National Science Foundation for its program directors and visiting scientists. The authors would like to thank Artur Andrzejak, Bill Bynum, Amy Dalal, Yong Guan, Brian Levine, Clay Shields, Wenting Tang, and Yong Yan for their valuable feedback. They would also like to thank Ludmila Cherkasova and John Sontag for their support at HP Labs. The comments from the anonymous referees are helpful and constructive.

REFERENCES

[1] Boeing log files, ftp://researchsmp2.cc.vt.edu/pub/boeing, 2003.
[2] D. Chaum, "Untraceable Electronic Mail Return Addresses, and Digital Pseudonyms," *Comm. ACM,* vol. 24, no. 2, pp. 84-88, Feb. 1981.
[3] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System: Design Privacy Enhancing Technologies," *Proc. Workshop Design Issues in Anonymity and Unobservability,* pp. 46-66, July 2000.
[4] R. Dingledine, M.J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Proc. Workshop Design Issues in Anonymity and Unobservability,* pp. 67-95, July 2000.
[5] P. Druschel and A. Rowstron, "PAST: A Large-Scale, Persistent P2P Storage Utility," *Proc. Eighth Workshop Hot Topics in Operating Systems,* May 2001.
[6] M.J. Freedman, E. Sit, J. Cates, and R. Morris, "Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer," *Proc. First Int'l Workshop Peer-to-Peer Systems,* Mar. 2002.
[7] http://www.freedom.net, 2003.
[8] E. Gabber, P. Gibbons, D. Kristol, Y. Matias, and A. Mayer, "Consistent, yet Anonymous, Web Access with LPWA," *Comm. ACM,* vol. 42, no. 2, pp. 42-47, Feb. 1999.
[9] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer, "How to Make Personalized Web Browsing Simple, Secure, and Anonymous," *Proc. Conf. Financial Cryptography,* pp. 17-31, Feb. 1997.
[10] Gnutella, http://gnutella.wego.com, 2001.
[11] Y. Guan, X. Fu, R. Bettati, and W. Zhao, "An Optimal Strategy for Anonymous Communication Protocols," *Proc. 22nd Int'l Conf. Distributed Computing Systems,* pp. 257-266, July 2002.
[12] Napster, http://www.napster.com, 2003.
[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *Proc. ACM SIGCOMM,* pp. 161-172, Aug. 2001.
[14] M.K. Reiter and A.D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Trans. Information and System Security,* vol. 1, no. 1, pp. 66-92, Nov. 1998.
[15] RSAREF20, http://tirnanog.ls.fi.upm.es/Servicios/Software/ap_crypt/disk3/rsaref20.zip, 1994.
[16] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking,* Jan. 2002.
[17] V. Scarlata, B.N. Levine, and C. Shields, "Responder Anonymity and Anonymous Peer-to-Peer File Sharing," *Proc. Ninth Int'l Conf. Network Protocols,* pp. 272-280, Nov. 2001.
[18] A. Serjantov, "Anonymizing Censorship Resistant Systems," *Proc. First Int'l Workshop Peer-to-Peer Systems,* Mar. 2002.
[19] A. Shamir, "How to Share a Secret," *Comm. ACM,* vol. 22, no. 11, pp. 612-613, Nov. 1979.
[20] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P^5: A Protocol for Scalable Anonymous Communication," *Proc. IEEE Symp. Security and Privacy,* May 2002.
[21] C. Shields and B.N. Levine, "A Protocol for Anonymous Communication over the Internet," *Proc. Seventh ACM Conf. Computer and Comm. Security,* pp. 33-42, Nov. 2000.
[22] I. Stoica, R. Morris, D. Karger, M.F. Kasshoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM,* pp. 149-160, Aug. 2001.
[23] A.B. Stubblefield and D.S. Wallach, "Dagster: Censorship-Resistant Publishing without Replication," Technical Report TR01-380, Dept. of Computer Science, Rice Univ., July 2001.
[24] P.F. Syverson, D.M. Goldschlag, and M.G. Reed, "Anonymous Connections and Onion Routing," *Proc. 1997 IEEE Symp. Security and Privacy,* pp. 44-53, 1997.
[25] M. Waldman and D. Mazi, "Tangler: A Censorship-Resistant Publishing System Based on Document Entanglements," *Proc. Eighth ACM Conf. Computer and Comm. Security,* pp. 126-135, 2001.
[26] M. Waldman, A.D. Rubin, and L.F. Cranor, "Publius: A Robust, Tamper-Evident, Censorship-Resistant Web-Publishing System," *Proc. Ninth USENIX Security Symp.,* pp. 59-72, Aug. 2000.
[27] M. Wright, M. Adler, B.N. Levine, and C. Shields, "An Analysis of the Degradation of Anonymous Protocols," *Proc. Ninth Ann. Symp. Network and Distributed System Security,* Feb. 2002.
[28] L. Xiao, S. Chen, and X. Zhang, "Dynamic Cluster Resource Allocations for Jobs with Known and Unknown Memory Demands," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 223-240, Mar. 2002.
[29] L. Xiao, X. Zhang, and Z. Xu, "A Reliable and Scalable Peer-to-Peer Web Document Sharing System," *Proc. Int'l Parallel and Distributed Processing Symp.,* Apr. 2002.

Li Xiao received the BS and MS degrees in computer science from Northwestern Polytechnic University, China, and the PhD degree in computer science from the College of William and Mary in 2002. She is an assistant professor of computer science and engineering at Michigan State University. She was a recipient of a USENIX Fellowship for her PhD dissertation research from 2001 to 2002. Her research interests are in the areas of distributed and Internet systems, system resource management, and design and implementation of experimental algorithms. She is a member of the ACM and the IEEE.

Zhichen Xu received the PhD degree in computer sciences from the University of Wisconsin, Madison, and the BS degree in computer sciences from Fudan University, Shanghai, China. He is a research scientist at Hewlett-Packard Laboratories working on network and storage system-related projects. Besides his current efforts, his other research interests include program safety, software environment, and parallel and distributed systems. He is a member of the IEEE.

Xiaodong Zhang received the BS degree in electrical engineering from Beijing Polytechnic University in 1982, and the MS and PhD degrees in computer science from the University of Colorado at Boulder in 1985 and 1989, respectively. He is the Lettie Pate Evans Professor of computer science and the department chair at the College of William and Mary. He was the program director of advanced computational research at the US National Science Foundation from 2001 to 2003. He is a past editorial board member of *IEEE Transactions on Parallel and Distributed Systems*, and currently serves as an associate editor of *IEEE Micro*. His research interests are in the areas of parallel and distributed computing and systems, and computer architecture. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.