Mutual Anonymity Protocols for Hybrid Peer-to-Peer Systems [*]

Li Xiao
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824
lxiao@cse.msu.edu

Zhichen Xu
Hewlett-Packard Laboratories
Hewlett-Packard Company
Palo Alto,CA 94304
zhichen@hpl.hp.com

Xiaodong Zhang
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187
zhang@cs.wm.edu

Abstract

In a hybrid peer-to-peer (P2P) system, some operations are intentionally centralized, such as indexing of peers' files. We present several protocols to achieve mutual communication anonymity between an information requester and a provider in a hybrid P2P information-sharing environment with trusted index servers such that neither the requester, nor the provider can identify each other, and no other peers can identify the two communicating parties with certainty. Some existing protocols provide solutions to achieve mutual anonymity in pure P2P systems without any trusted central controls. Compared with two representative protocols, our proposed mutual anonymity protocols improve efficiency by utilizing trusted third parties and aiming at both reliability and low-cost. We show that with some limited central support, our protocols can accomplish the goals of anonymity, efficiency, and reliability. We have evaluated our techniques in a browser-sharing environment. We show that the average increase in response time caused by our protocols is trivial, and these protocols show advantages over existing protocols in a hybrid P2P system.

1. Introduction

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

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 non-producer peers.

Depending on 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.

Our goal is to achieve mutual anonymity between the initiator and responder with high efficiency in hybrid P2P systems utilizing trusted index servers (e.g., Napster [10], and browser-aware proxies [21]). 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, significantly 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.

We have evaluated our techniques in a browser-sharing environment. We show that the average increase in response time caused by our protocols is trivial, and these protocols show advantages over existing protocols in a hybrid P2P system.

2. Related Work and Motivation

The related work includes existing protocols for the three types of anonymity. We have paid a special attention to the work on mutual anonymity, which has motivated us to develop new protocols.

[*]This work is supported in part by the U.S. National Science Foundation under grants CCR-9812187, EIA-9977030, CCR-0098055 , and by a USENIX Research Scholarship.

2.1. Publisher and Sender Anonymity

Publisher Anonymity: In order to protect a publisher peer, many systems provide 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 [20] splits the symmetric key used to encrypt and decrypt a document into n shares using Shamir secret sharing and store 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 [14] 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 [19] and Dagster[17] 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 [8] and Lucent Personalized Web Assistant (LPWA) [7] 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 [18], 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 [11] forms a covert path in such a way that the next node is randomly selected by its previous node. Hordes [16] applies a similar technique used in Crowd, but it uses multicast services to anonymously route the reply to the initiator. Freedom [6] and Tarzan [5] 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) [15], and Anonymous Peer-to-peer File Sharing (APFS) [13].

Paper [15] first proposes 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 a 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 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. Thirdly, 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.

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 $(i = 1, 2, ...)$ to represent a peer. For conciseness of the presentation, we assume there is only one index server. Section 4 discusses how multiple index servers will be involved in order to scale a P2P system.

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 $p_0, p_1, ..., p_k$ 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 $(p_0, (p_1 ...(I, fakemix)K_{p_k} ...)K_{p_0})K_R$. 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. $(p_1 ...(I, fakemix)K_{p_k} ...)K_{p_0}$. 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.

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. Figure 1 shows an example with two middle nodes.

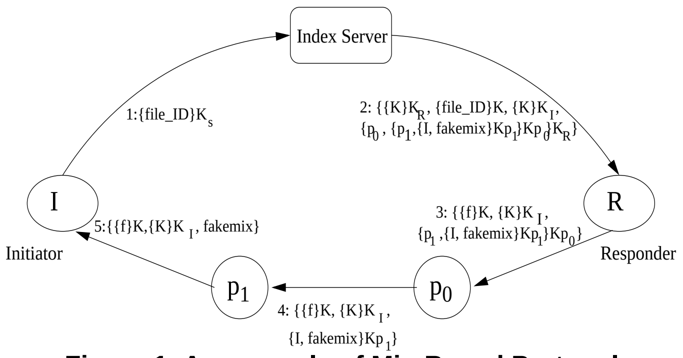


Figure 1. An example of Mix-Based Protocol

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 *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 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 random 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, 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.

$$R \rightarrow p_0 : \{\ldots\{n\}K_{p_{j_0}}\ldots\}K_{p_{j_{i+1}}}, \{f\}K, \{K\}K_I$$

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

Figure 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.

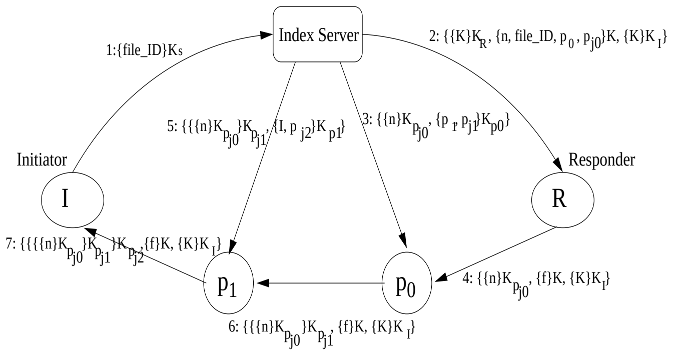

Index Server
1:{file_ID}Ks
2: {{K}K_R, {n, file_ID, p_0, p_j0}K, {K}K_I}
5: {{{n}K_pj0}K_pj1, {I, p_j2}K_p1}
3: {{n}K_pj0, {p_1, p_j1}K_p0}
Initiator
Responder
I
R
7: {{{{n}K_pj0}K_pj1}K_pj2, {f}K, {K}K_I}
p1
p0
4: {{n}K_pj0, {f}K, {K}K_I}
6: {{{n}K_pj0}K_pj1, {f}K, {K}K_I}

Figure 2. Example of Center-Directing Protocol

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 sub-tables and distributed to peers (encrypted with their public keys). The sub-table 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 sub-table.

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. Sub-Tables

Peer1	
L1	7
…	…
…	…

Peer2	
L8	3
…	…
…	…

Peer3	
L8	0
L4	**4**
…	…

Peer4	
L3	6
L4	**0**
…	…

Table 1 shows an example path table with 4 options for each peer. Table 2 shows some sub-tables derived from Table 1. In this example, each path option has two middle nodes. The number of middle nodes is not fixed in our design. It has already been shown that variable path-length strategies perform better than fix-length strategies[9]. Assuming that the index server needs to prepare a path from node 5 to 0, it can select among 4 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 sub-tables.

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

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

$Step$ 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 sub-table of p_i. A persistent connection is set between p_i and p_{i+1}.

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

$Step$ 4: A message is sent along the persistent connections from R to I. We use $-l \rightarrow$ to represent the persistent connection identified by the label l.

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

$Step$ 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 sub-table distribution. The overhead comes from the spaces for storing the path table and sub-tables 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.

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, 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 load sharing schemes 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 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.

5. 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 how different parties can 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 probability that there are k middlemen is $p(k)$, the probability that the responder makes the right bet is $p(0)$.

A middle node: We consider two cases: In the first case the middleman makes a random guess, because the only thing she can be sure about is that she is not the initiator. In this case, the probability it makes a correct guess is $\frac{1}{n-1}$. In the second case, the middleman bets that the peer to which it sends the message is the initiator. If there are k middlemen, only one of the k middlemen will make a correct bet. The probability that a middleman can make the correct bet is $\frac{1}{n-2}\sum_{k=1}^{n-2}\frac{p(k)}{k}$, and $p(k)$ is the probability that there are k middlemen.

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 what is communicated.

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 middlemen (for the shortcut-responding protocol).

Cooperating Peers: We consider cases where at least two middle nodes cooperate, and assume that neither the responder nor the initiator is involved. 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 that they make the right guess is $\frac{1}{n-k}$, because all peers other than the k peers can be the initiator. If the collaborating peers were to make a bet, they can first eliminate all the peers that are communicating with peers that they know for sure is not the initiator. The worst scenario is when at least $m-1$ out of all m middle nodes are involved. Even in this case, these middle nodes only have $\frac{1}{2}$ probability of correctly guessing that there is only one communication is conducted. The probability for them to correctly bet the identity of the initiator is $\frac{1}{2}$.

Our protocols keep the same anonymity degree with P^5 and APFS while achieving high efficiency shown in the next section.

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 [21]. 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 March 5, 1999). There are 3996 and 3659 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.

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, we can see that the amounts of data transfer times spent for communications

[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.

among clients on both traces are very low, which is shown in Figure 3.

6.2. Overhead of DES and RSA

The source programs of DES and RSA are obtained from [12]. The machine we used for the experiments is a PC with a 1000 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 are used in our simulations to calculate the overheads of DES and RSA.

6.3. Comparisons of Protocols

We have shown how we measure the data transfer times and the costs of DES and RSA operations. Here we compare the accumulated overheads of the protocols. Figure 3 compares the total increased response times and their breakdowns for the protocols using the "Boeing March 4 trace" and "Boeing March 5 trace" with 2 and 5 clients acting as middle nodes.

The performance results in Figure 3 show that center-directing and label-switching protocols generate very low overhead, while mix-based protocol has 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 3.7 ms when we use 5 middle nodes for both traces. The two protocols with lower overhead only increase the response time to about 2.7 ms when 5 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. 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.

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. Paper [9] shows that the anonymity degree may not always monotonically increase as the length of communication path increases.

7. Conclusion

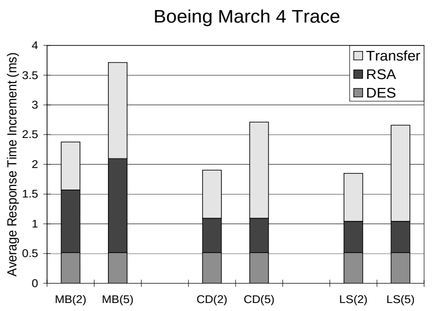
Providing a reliable and efficient anonymity protection among peers is highly desirable in order to build a scalable and secured P2P systems. In this paper, we have presented several protocols to achieve mutual anonymity in a hybrid P2P file-sharing environment. Our protocols take advantage of the existence of trusted third parties to improve efficiency and reliability, and use them to prepare the covert paths for anonymous communications.

The 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 co-exist 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 pre-defined 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.

To a great extent, the performance and robustness of a P2P system depend on the capacity of trusted servers and on 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.



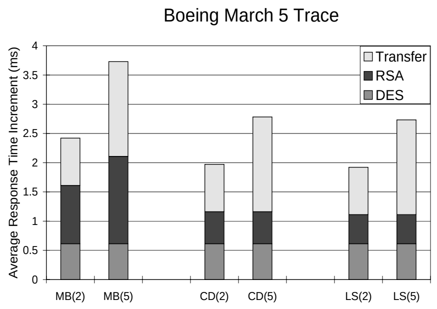


Figure 3. Breakdown of data transfer and protocol overhead with 2 and 5 middle nodes for Boeing March 4 trace (left) and Boeing March 5 Trace (right). $MB(k)$ represent mix-based protocol with k middle nodes. Similarly, CD, and LS represent center-directing, label-switching, respectively.

Acknowledgment: The work is a part of an independent research project sponsored by the National Science Foundation for its program directors and visiting scientists. We appreciate Bill Bynum, Amy Dalal, Brian Levine, Clay Shields, Wenting Tang, and Yong Yan for their valuable feedback. We thank John Sontag and Ludmila Cherkasova for their support. The comments from the anonymous referees are helpful and constructive.

References

[1] Boeing log files. ftp://researchsmp2.cc.vt.edu/pub/boeing.

[2] D. Chaum, “Untraceable Electronic Mail Return Addresses, and Digital Pseudonyms”, *Communications of the ACM*, 24, 2, Feb.1981, pp.84-88.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: a distributed anonymous information storage and retrieval system. Design Privacy Enhancing Technologies”, *Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, ed. by H. Federrath, Springer-Verlag, 2001, pp.46-66.

[4] R. Dingledine, M. J. Freedman, and D. Molnar, “The Free Haven Project: Distributed Anonymous Storage Service”, *Workshop on Design Issues in Anonymity and Unobservability*, LNCS 2009, ed. by H. Federrath, Springer-Verlag, 2001, pp.67-95.

[5] M. J. Freedman, E. Sit, J. Cates, and R. Morris, ”Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer”, *Proceedings of the 1st International Workshop on Peer-to-peer Systems*, Cambridge, MA, March, 2002.

[6] Freedom, http://www.freedom.net/

[7] E. Gabber, P. Gibbons, D. Kristol, Y. Matias, and A. Mayer, “Consistent, yet anonymous, Web access with LPWA”, *Communications of the ACM*, 42, 2, February 1999, pp.42-47.

[8] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer, “How to make personalized Web browsing simple, secure, and anonymous”, *Proceedings of 1997 Conference on Financial Cryptography*, Springer-Verlag, New York, NY, 1997, pp.17-31.

[9] Y. Guan, X. Fu, R. Bettati, and W. Zhao, “An optimal strategy for anonymous communication protocols”, *Proceedings of the 22nd International Conference on Distributed Computing Systems*, (ICDCS’2002), July 2002, pp.257-266.

[10] Napster, http://www.napster.com.

[11] M. K. Reiter, and A. D. Rubin, “Crowds: Anonymity for Web Transactions”, *ACM Transactions on Information and System Security*, 1,1, November 1998, pp. 66-92.

[12] RSAREF20, http://tirnanog.ls.fi.upm.es/Servicios/software/ap/crypt/disk3/rsaref20.zip

[13] V. Scarlata, B. N. Levine and C. Shields, “Responder Anonymity and Anonymous Peer-to-Peer File Sharing”, *Proceedings of the 9th International Conference on Network Protocols* (ICNP 2001), November, 2001.

[14] A. Serjantov, ”Anonymizing Censorship Resistant Systems”, *Proceedings of the 1st International Workshop on Peer-to-peer Systems*, Cambridge, MA, March, 2002.

[15] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, ”P^5: A Protocol for Scalable Anonymous Communication”, *Proceedings of 2002 IEEE Symposium on Security and Privacy*, May 2002, pp.58-70.

[16] C. Shields and B. N. Levine, “A Protocol for Anonymous Communication Over the Internet”, *Proceedings of 7th ACM Conference on Computer and Communication Security* (CCS’2000), November 2000, pp.33-42.

[17] A. B. Stubblefield and D. S. Wallach, ”Dagster: Censorship-Resistant Publishing Without Replication”, Technical Report TR01-380, Department of Computer Science, Rice University, July 2001.

[18] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, “Anonymous Connections and Onion Routing”, *1997 IEEE Symposium on Security and Privacy* (S&P’97), pp.44-53.

[19] M. Waldman, D. Mazi, ”Tangler: a censorship-resistant publishing system based on document entanglements”, *Proceedings of the 8th ACM conference on Computer and Communications Security*, 2001, pp.126 - 135.

[20] M. Waldman, A. D. Rubin, and L.F. Cranor, ”Publius: A Robust, tamper-evident, censorship-resistant Web-publishing system”, *Proceedings of the 9th USENIX Security Symposium*, August 2000, pp.59-72.

[21] L. Xiao, X. Zhang, and Z. Xu, “A reliable and scalable peer-to-peer Web document sharing system”, *Proceedings of 2002 International Parallel and Distributed Processing Symposium*, (IPDPS’2002), Fort Lauderdale, Florida, April 15-19, 2002.