A Mutual Anonymous Peer-to-peer Protocol Design

Jinsong Han[1], Yunhao Liu[1], Li Xiao[2], Renyi Xiao[1,3], Lionel M. Ni[1]

[1]*Dept. of Computer Science, Hong Kong University of Science and Technology, Hong Kong*
[2]*Dept. of Computer Science & Engineering, Michigan State University, East Lansing, MI48824, USA*
[3]*National Natural Science Foundation, Beijing, China*
{jasonhan, liu, renyi, ni}@cs.ust.hk; lxiao@cse.msu.edu

ABSTRACT

Peer-to-Peer (*P2P*) *computing has become a popular application model because of its easy resource sharing pattern and powerful query search scheme. However, decentralized P2P architecture is scarcely seen to deploy the anonymity on its peers. In this paper, we propose a mutual anonymity protocol, called Secret-sharing-based Mutual Anonymity Protocol* (*SSMP*)*, for decentralized P2P systems. SSMP employs Shamirs' secret sharing scheme to allow peers to issue queries and responders to deliver requested files anonymously. Compared with previous designs, SSMP achieves mutual anonymity in P2P systems with a high degree of anonymity and a low cryptography processing overhead. We evaluate SSMP by comprehensive simulations.*

1. Introduction

The Internet has been developed into a globally discrete information sharing system over the past few years [10, 12, 18, 19, 25]. While the network users' privacy requirements have become increasing urgent, the anonymity issues in some particular systems have not yet been fully addressed. The most important protocol of the Internet, TCP/IP, pays less attention on privacy than other performance issues such as efficiency and scalability. The easy track feature of TCP/IP makes private and secure data vulnerable when being transferred in the open Internet environment.

Under certain circumstances, network users may require different types of anonymity [4]. Anonymity can be divided into three types: resistant-censorship (or publishing anonymity); initiator or responder anonymity; and mutual anonymity (giving both the initiator and responder anonymity). Strictly defined, mutual anonymity is made up of three parts: an anonymous initiator, an anonymous responder and the anonymous communication between these two units. Most previous work provides mutual anonymity protocols for clients with the help of trusted agents or proxy servers.

Recently, peer-to-peer (P2P) file sharing applications, such as Napster, Gnutella and KaZaA [1-3], have made it convenient to search and obtain desired contents from the Internet. In P2P systems, peers join and leave the network freely and frequently. Each peer performs as both a content provider and a consumer.

There are two major architectures for P2P systems: centralized and decentralized. A centralized P2P system, such as Napster, is vulnerable to denial of service attacks [29, 44] and suffering from a single point of failure. Decentralized P2P systems, which have the advantages of high fault-tolerance, sufficient autonomy, and flexible scalability, are widely deployed.

This paper focuses on decentralized and unstructured P2P systems [10, 12, 16, 20, 26-28, 38, 41]. All participants in such a P2P system communicate only with their neighbors. Theoretically, no peer has any knowledge of other peers two or more hops away. Since a query message does not include the IP address of its query peer, current P2P systems achieve a certain degree of anonymity, which is incomplete based on following two observations.

First, a peer's identity is exposed to all its neighbors. Some malicious peers can acquire information easily by monitoring packet flows, distinguishing packet types, (e.g., the Query Hits type message [5]), and analyzing the TTL value of these queries. In this way, initiators and responders are not anonymous to their neighbors and P2P systems fail to provide anonymity in each peer's local environment.

Second, in the query and reply packet transfer path, there are high risks that the identities of both the initiators and responders are exposed. In an untrustworthy public network, when the files are transferred in a plain text model, the contents of the files also help the attackers on the path guess the identities of the communication parties.

This work was partially supported by Hong Kong RGC DAG 04/05.EG01, RGC Grants HKUST 6264/04E, and US National Science Foundation under grant CCF-0325760.

Therefore, current P2P systems cannot provide anonymity guarantees. In this paper, we propose a mutual anonymity protocol in decentralized P2P systems, called Secret-sharing-based Mutual Anonymity protocol (SSMP). SSMP provides both initiator and responder anonymity, and communication security as well. In SSMP, we employ the idea of a secret sharing scheme to guarantee anonymous query issue, and introduce the information dispersal algorithm (IDA) [30] together with the onion routing method [39] to achieve a complete reply-confirm interaction between initiators and responders. The main advantages of this protocol include a high degree of anonymity and low cryptographic overheads than those in previous mutual anonymity protocols. We evaluate SSMP by trace-driven simulations, and compare its performance with existing approaches.

The rest of this paper is organized as follows. In Section 2, we introduce previous work on anonymity. In Section 3, we discuss the SSMP design. In Section 4, we analyze the degree of anonymity and degree of security of the proposed SSMP. We provide our implementation experience and simulation results in Section 5. We conclude the work in Section 6.

2. Related Work

Many efforts have been made to acquire anonymity in P2P systems. Earlier work falls into two categories: anonymous publishing and anonymous communication.

Recently, there are two kinds of such approaches to provide publisher anonymity protocol. One employs hash to mark the key information of documents, such as Freenet [11]. The other one, such as those used in Publius [40], Freehaven [13], [34], GNUnet [23] and Gap [8] use Shamir's secret sharing like scheme to either split a symmetric key or break the file into *n* shares to achieve the goal of anonymous file sharing.

Anonymous communication includes initiator and responder anonymity and anonymous data transferring. In MorphMix [32] and Onion [17], initiator can determine an anonymous path in advance to hide some identification information. Instead of building a path by an initiator, Crowds [31] and Hordes [37] let middle nodes select the next hop on the path and deploy multicast trees by themselves. Freedom [7], Tarzan [16] and Tor [14] are implemented based on Onion Routing to provide anonymous services.

Some studies, such as Peer-to-Peer Personal Privacy Protocol (P^5) [36], Anonymous Peer-to-Peer File Sharing (APFS) [33], and Shortcut-responding Protocol [43], proposed to provide mutual anonymity in P2P systems. The basic idea of P^5 is to let all participants in the channel send fixed length encrypted packets at a fixed rate as if all participants are in a logic ring. This protocol introduces noise packets to maintain a fixed communication rate to confuse the traffic analyzers or attackers. Meanwhile, P^5 refines the broadcast channel size and builds a hierarchy overlaid spanning tree to keep the communication scalable. P^5 users can join different groups based on the tradeoff between anonymity degree and the communication bandwidth and reliability. As P^5 assumes that every initiator knows the public keys of all possible responders, it cannot be directly employed in P2P networks.

APFS is designed for a decentralized system, like Gnutella. Some coordinator nodes act as a superior peer and maintain a list of all the peering nodes. Some peers in these lists volunteer to issue queries for others. When an initiator queries a coordinator for available servers, the coordinator returns a list of current servers. The initiator then contacts some servers to send the request and receive replies from the servers. It finally sends the request and receives the reply with the help of the tail node of the matching responder. All communications of this framework are based on the onion path to guarantee the anonymity and hence no centralized authority exists in this system.

In Shortcut-responding Protocol [43], the initiator establishes an onion-based reply block, called re-mailer, before sending each query. Such a re-mailer is in fact an anonymous return path. Each peer that receives the query determines whether devote itself as a query agent peer in a probability of *pv*. If a peer acts as the query agent for the initiator, it floods this query into P2P systems. Upon requests, a responder builds another onion path to anonymously send the file to the query agent peer. The query agent peer delivers the file along the returning path to the initiator. Although reducing the length of the return path, this approach does not consider the reply-confirm procedure between the initiator and the responder. By adding this feature to the design of SSMP, we further show that SSMP outperforms Shortcut-responding Protocol in efficiency and security.

3. Design of SSMP

In this section, we first introduce the secret sharing scheme and an information dispersal algorithm (IDA), which are the basic algorithms used in SSMP. We then present the details of the SSMP protocol.

3.1 Secret Sharing Scheme

The motivation of the secret sharing scheme is to avoid a single point of failure during the key maintenance procedure. In this scheme, a secret sender (or dealer) distributes secret *s* to *n* participants. It requires no less than *k* ($k<n$) players of *n* participants to recover the secret *s*.

One of the most popular secret sharing schemes is Shamir's secret sharing scheme [35]. The mathematical basis of this scheme is as follows. Given *k* points on the

Table 1: Switch Table Example

Sequence Number	IP of neighbors	Contents of Shares
*sq*1	From *IP*2	*h*, K_O
*sq*2	To *IP*3	*h*, K_O
…	…	…
*sq*5	From *IP*7	Normal query
…	…	…

plane $(x_1,y_1),...,(x_k,y_k)$, if all x_i's are distinct, there exists a unique polynomial f of degree $\leq k-1$, $f(x_i) = y_i$ for all i.

The constructive method is that given these k points, f can be recovered by using Lagrange's interpolation formula. Normally, all these hold in a field Z_p, where p is a prime.

Based on these discussions, we describe Shamir's scheme as follows.

To share secret s among n entities and ensure that no less than k participants are required to recover the secret, dealer D creates a random polynomial $f(x)$ of degree $k-1$:

$$f(x) = a_0 + a_1 x + ... + a_{k-1} x^{k-1}$$

This polynomial is constructed over a finite field Z_p, and the coefficient a_0 is the secret s. All other coefficients are random elements in the field; where the field is known to all participants. Dealer D publicly chooses n random distinct evaluation points: x_i, and secretly distributes the $share_i(s) = (x_i, f(x_i)), i=1..n$ to each player P_i. We can prove the theorem: the secret s can be reconstructed from every k-share subset. Using Langrange *formula*, given k points (x_i,y_i), $i = 1..k$, we have

$$f(x) = \sum_{i=1}^{k} y_i \prod_{j=1, j\neq i}^{k} \frac{x - x_j}{x_i - x_j} (\mathrm{mod}\, p)$$

Thus

$$s = f(0) = \sum_{i=1}^{k} y_i \prod_{j=1, j\neq i}^{k} \frac{-x_j}{x_i - x_j} (\mathrm{mod}\, p)$$

The complexity of Lagrange interpolation is $O(klog_2k)$.

3.2 Information dispersal algorithm (IDA)

IDA was first introduced by Rabin [30]. Similar to Shamir's scheme, a (m, n) IDA intends to distribute information s among n processors. Meanwhile, the recovery of the information is available if the collection of shares is up to m $(1<m<n)$. The difference between IDA and Shamir's scheme is that the length of each distributed fragment of IDA is not $|F|$, but $|F|/m$, where $|F|$ is the file size. Hence, IDA is more space efficient. IDA is often employed together with the erasure code scheme [30]. As the basic IDA does not provide protection from malicious parties or security for the data, we replace it with a secret information dispersal algorithm *SIDA*[22].

3.3 SSMP Protocol

Since Shamir's secret sharing scheme provides prefect secrecy, we use it in SSMP to distribute the DES key. We employ *SIDA* to split the requested files for the purpose of space efficiency.

The basic idea of SSMP is as follows. When a peer is ready to issue a query, it first distributes the requested file ID f into n pieces of secret shares using Shamir's secret sharing scheme. It then sends out the split secret shares to some random neighbors. Its neighboring peers flood the fragments with a certain probability. When any peer, P_i, $1<i<n$, collects k or more shares, the plain query information is recovered, and P_i floods the query message into the P2P system. Once the peer is able to provide the requested file, it creates two onion paths: one is for sending the reply information back to the P_i; and the other one is built as an anonymous path for the initiator to return the confirmation packets. Eventually the chosen responder peer that receives the confirmation will split the file into m fragments using *SIDA* scheme and delivers the requested data back to the initiator.

Before going into details of SSMP, we first introduce some notations used in this paper. Let f denote the desired file name or id, O denote the initiator of a query, r_i denote O's neighboring peers, R denote the query responder, P_i denote peers in the P2P system, and sq denote the sequence number to mark queries. We use $A \rightarrow B: M$ to represent A sending a message M to B .We use K_X to represent the public key of peer X and K to denote the DES key. We define $\{M\}K_X$ as encrypting the message M with the public key K_X (RSA) of peer X, and $E_K(M)$ as the ciphertext of M under the symmetric key K (DES). Let $D_K(\cdot)$ denote the corresponding decryption transformation.

In SSMP, we modify Gnutella 0.6 [5] protocol by adding a local switch table to each servent to record the packet's information received from or sent to the neighbors. Thus packets can be delivered in correct directions. In addition, a timeout scheme is introduced to keep the table size scalable. When a peer joins, it automatically establishes a local switching table like the example shown in Table 1.

The SSMP is conducted in the following steps.

Step 1: The initiator O randomly selects a list of neighboring peers, $r_1, r_2... r_n$. It generates a random DES key T, which is used to encrypt f.

Distribution method:

– Select random key T and encrypt f as $e = E_T(f)$.

– Partition e into n fragments using information dispersal algorithm [30], $e_1...e_n$.

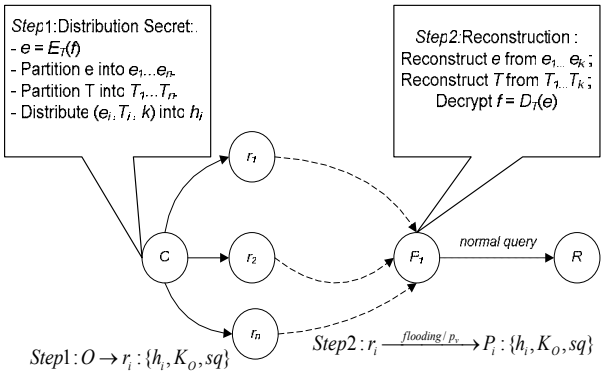

Figure 1: Share flooding

– Partition T into n shares using Shamir (k, n)-method, $T_1 \ldots T_n$.

– Distribute (e_i, T_i, k) to n pieces h_i.

Meanwhile, O creates a sequence number sq to identify its query. In addition, O creates a pair of RSA keys, and we use K_O to denote its public key. Peer O randomly chooses neighboring peers r_i, and sends h_i, $i = 1..n$ to them, respectively. When sending these messages, O embeds K_O in these packets: $O \rightarrow r_i: \{h_i, K_O, sq\}$.

Step 2: When O's neighbors receive a new query share, they flood these shares in probability p_v. If not flooding, they simply forward query shares to one of their neighboring peers except O.

$$r_i \xrightarrow{flooding\ /\ p_v} P_j : \{h_i, K_O, sq\}$$

For both flooding and a single-direction-forwarding pattern, r_i stores the received and forwarding information of each query in its local switch table.

Once a peer receives a share query, it compares sq with its locally received share query in the switch table. If the share belongs to an existing query group, it stores this share information to the same label subset; otherwise it establishes a new subset with the new sq and put this share in. During certain number of hops, different share queries labeled same sq would be gathered in one or more peers. Our later experimental results show that carefully selecting the value of p_v would lead to gathering satisfied amount of the shares. Without loss of generality, we suppose P_1 is one of the peers that collect enough number of shares of the query to recover f.

Reconstruction method:

– Given $k < n$ shares $h_i = (e_1, T_1) \ldots (e_k, T_k)$ and k.

– Reconstruct e from $e_1 \ldots e_k$.

– Reconstruct T from $T_1 \ldots T_k$.

– Decrypt $f = D_T(e)$.

When P_1 obtains the recovered f, it floods the query in plain text. Figure 1 illustrates step 1 and 2.

Step 3: When a peer R is able to provide the requested file f, it builds two anonymous paths in advance based on bidirectional onion protocol.

Forwarding path:

$$FP = \{X, \{Y, \ldots \{P_1, reply, RP, sq\} K_{P_1}, \ldots\} K_Y\} K_X$$

Returning path:

$$RP = \{U, \{V, \ldots \{Z\{R, fixmix\} K_Z\}, \ldots\} K_V\} K_U\} .$$

The reply message includes an index of desired files. The responder delivers the packet through FP to P_1.When the reply packet reaches P_1, P_1 first decrypts the cipher and gets sq and file index. It then checks sq in its local switch table. Before sending it back to peer O, P_1 should organize all received replies to a single packet and use K_O to encrypt it. Then P_1 randomly chooses a return peer from its switch label according to sq. In next hop, the receiver checks sq in its local switch table. If the reply packet does not belong to it, it forwards this packet again to the next randomly chosen peer marked with sq, until the packet reaches peer O.

Step 4: There could be more than one peer who can recover f. At the same time, peer O may receive more than one reply from certain peers delivered by those P_i. Peer O picks one as a responder, R. Peer O then creates a list, L, for all available middle P_i, where $L = \{P_1 \ldots P_m\}$. In addition, suppose it selects a number $k' < m$, as the *SIDA* threshold for splitting the requested file and puts k' in the confirmation packet. Peer O sends the packet embedded with the list L, k' and K_O to the first peer which is appointed by the responder R on its onion return path (RP).

Step 5: R gets the request-confirm packet from the last RP onion peer. It makes use of *SIDA* as follows.

Distribution method:

– Select a random symmetric key K and encrypt f with K: $e' = E_K(f + hash(f))$

– Encrypt key K with key K_O to get T'

– Partition e' into m fragments using (k', m) information dispersal algorithm and achieve $e'_1 \ldots e'_m$. ($m = |l|$).

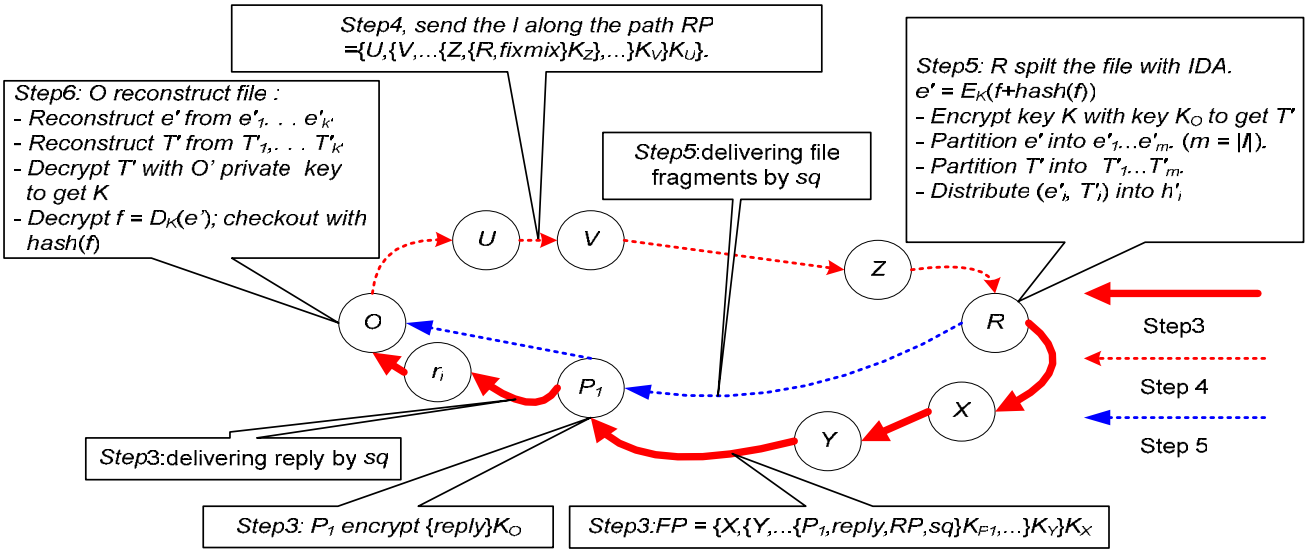

Figure 2: Query reply, confirmation and file delivery

– Partition T' into m shares using (k', m) Shamir's scheme to get $T'_1 \ldots T'_m$.
– Distribute (e'_i, T'_i) to participant h'_i

Note that we assume the asymmetric and symmetric algorithms are all polynomial indistinguishable. R finally obtains m split data fragment packets h'_i for m peers in L, and marks the packet as a data type and labels it sq. The entire information of list L is put in every packet as well. After these steps R randomly chooses its m neighbors (if R's neighbors are less than m, R can send some packets to the same neighbor), and send the above packets.

Each peer who receives a packet randomly selects a peer d from L and forwards the packet to it. Peer d then delivers the packet to the next peer marked sq. The next peer would do the same thing until the data packet reaches some members of L. This peer in L will deliver the packet to peer O as in step 3.

Step 6: Peer O keeps checking on the number of sq-marked data packets it receives.

Reconstruction method:
– Given $k' < m$ shares (e'_1, T'_1, k'),…, ($e'_{k'}$, $T'_{k'}$, k')
– Reconstruct e' from $e'_1 \ldots e'_{k'}$
– Reconstruct T' from $T'_1, \ldots T'_{k'}$
– Decrypt T' with O' private key to get K
– Decrypt f = $D_K(e')$; checkout the file with $hash(f)$

Figure 2 illustrates query reply, confirmation and file delivery procedure.

4. Discussions

In this section, we analyze the degree of anonymity of SSMP, and discuss the degree of security of the whole protocol.

4.1 Anonymity analysis

We first analyze the degree of anonymity of SSMP from different parties.

The initiator: The probability for an initiator to randomly guess the responder's identity is $1/(n-1)$, where n is the number of P2P peering nodes, normally millions in real systems. The reply of the query is forwarded by the agent peers. If the responder is not the agent peer, its identity is hidden by the onion path. With the onion path, the expected number of path reformations required for c attackers to determine the initiator out of n participants is $O((n/c)^l)$, where l is the length of the path between the initiator and responder [42].

The responder: As a responder, the probability for it to correctly guess the initiator's identity is also $1/(n-1)$. Actually all the queries it receives are coming from P_i s. It has no details of the share flow situation when the P2P system is in large scale.

The middle nodes: In the query-flooding path, the middle nodes can be divided into two groups: one includes the peers who receive the shares. The other group includes the remaining peers except the initiator and responder. For the first group, its members make a random guess on who the initiator or the responder are will have probability of $1/(n-1)$. Let $p_i(k)$ denote the probability of existing k peers between the initiator and peer x . Note that middle peers mean all such peers that have fewer

hops from the initiator to themselves than x does. S is a subset of the whole P2P system in which the peers get their shares from the initiator. The probability that x guesses right whether the peer from whom x obtains the share is the initiator is:

$$\frac{1}{|S|-1}\sum_{k=1}^{|S|-1}\frac{p_i(k)}{k}.$$

On file forwarding path, the random guess probability that a certain peer is the initiator is $1/(n\text{-}2)$. This is because files are divided into fragments when being transferred. Even though an attacker could recover a file from the shares, it cannot decrypt the content from the cipher text without the key. The probability that the guess is right will be:

$$\frac{1}{n-2}\sum_{k=1}^{n-2}\frac{p_i(k)}{k},$$

where k is the number of peers on data packets transferring path. The probability for the attacker to guess the correct responder is similar.

In current P2P systems with millions of nodes, all the exposed probabilities in the above cases become very small.

4.2 Security analysis

SSMP deploys encryption methods both in query flooding and file transferring to achieve information security. Malicious peers may cooperate together to implement attacks. SSMP makes their attacks difficult by the following three steps.

1) Shamir's scheme protects the initiators' privacy in a prefect security.

2) The file split pattern of *SIDA* provides a computational security to responders.

3) The cryptography mechanism provides the unlinkability security for transferring data.

Definition 1: Perfect secrecy is when, for all possible cryptograms, the posteriori probabilities are equal to the a priori probabilities independent of the values of these.

Theorem 1: SSMP holds a perfect security in shares' distribution.

Proof: In (t, n) Shamir's scheme $(t < n)$, given any t shares, the polynomial is uniquely determined. Hence, the secret s can be computed. However, given $t\text{-}1$ or fewer shares, the secret can be any element in the field. Thus those shares do not supply any further information regarding the secret [21]. If S_0 is the secret, for every $k < t$, let $S_1, \ldots, S_k$ be any k shares then $Pr(S_0 \mid S_1, \ldots, S_k) = Pr(S_0)$. Indeed we claim that the share distribution procedure of SSMP holds perfect secrecy. ■

Definition 2: An (m, n) threshold scheme $(m < n)$ is computational security if for any two secrets S' and S'', for any $k < m$, the distributions on shares $D(S'; S'_1, \ldots, S'_k)$ and $D(S''; S''_1, \ldots, S''_k)$ introduced by the scheme are polynomial indistinguishable.

Definition 3: $E_0 = \{X_n\}$ and $E_1 = \{Y_n\}$ are polynomial indistinguishable if for every (probability) polynomial-time algorithm, A, and every $c > 0$ there exists an integer N such that for all $n \geq N$,

$$\left|\Pr(A(X_n)=1) - \Pr(A(Y_n)=1)\right| < \frac{1}{n^c}.$$

A distinguish algorithm is the one that successfully guesses the correct distribution with probability $½ + l^c$, where l is the distribution index.

Theorem 2: *SIDA* achieves computational security.

Proof: We assume that there exist secret s_1 and s_2, and an algorithm A can distinguish them from their space S_1 and S_2 with a high probability:

$$\Pr(a) = \left|\Pr(A(s_1/S_1)=1) - \Pr(A(s_2/S_2)=1)\right| > \frac{1}{2}$$

Then we construct another algorithm B that can breach the polynomial indistinguishable encryption algorithm E by distinguishing between $E(s_1)$ and $E(s_2)$ in the probability

$$\Pr(b) = \Pr\left|B(E(s_1)/E(S_1)) - B(E(s_2)/E(S_2))\right|.$$

If $E' = SIDA(E(s')) = e'_1 \ldots e'_n$ and $X \subset E', |X| < m$, we then get X_1 and X_2 from $E(S_1)$ and $E(S_2)$, and use A to guess whether the secret corresponds to s_1 or s_2. B can also output the same guess. If A guesses the correct result with a probability over ½, then B obtains the correct result with a significant probability,

$$\Pr(b) = \Pr\left|B(e'/S')\right| > \frac{1}{2}$$

However, we employ polynomial indistinguishable encryption algorithms in Section 3. Therefore, we conclude that first assumption is nonexistent. In fact, the encryption algorithms' security ensures that the $m\text{-}1$ fragments X give no more information than E. Further, there is absolutely no information to reveal K without more than $m\text{-}1$ key shares, as we proved in Theorem1. ■

In SSMP, we use RSA and DES as the practical secure algorithms. Some improved RSA-based schemes can eventually provide indistinguishability [24]. Therefore, SSMP is able to protect file information from malicious entities and provide unlinkability of initiator and responder.

5. Performance Evaluation

5.1 Performance Metrics

Our protocol performance metrics focus on four major parts: query scope, response time, security overhead, and traffic cost.

In a pure P2P system, the QoS of a search system depends on the number of peers being explored (queried),

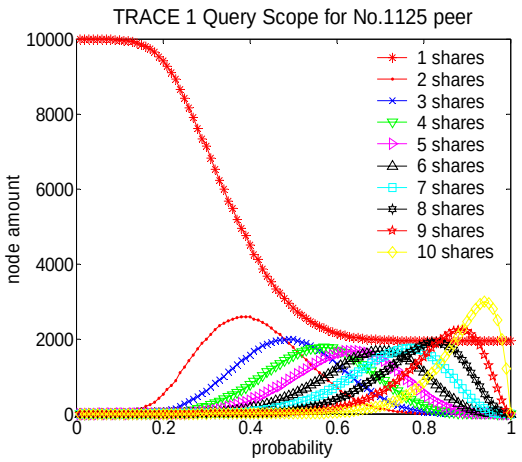

Figure 3 : A peer Query scope-Trace1

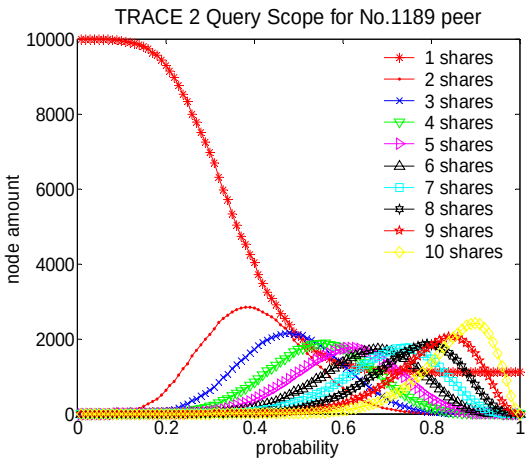

Figure 4 : A peer Query scope-Trace2

the response time, and the traffic overhead. In our protocol, we first present the distribution of shares gathering so that we can choose good parameters for the flooding probability and threshold of Shamir's scheme.

We then present the response time of our protocol, compared with the normal decentralized P2P query pattern and the anonymity protocols proposed in [43]. In addition, we present the security cost in our protocol and the traffic cost increased by SSMP.

Query scope is the number of peers that queries have reached in a search process. In SSMP, we define query scope as the number of peers which get enough shares to recover the secret in Shamir's secret sharing scheme. In a decentralized P2P system, the more peers reached means the higher likelihood that the requested file can be obtained. However it is useless to query the same content too many times, while this situation will occur if we do not control the number of the qualified query agents with the threshold of Shamir's scheme. For this reason, we show the relationship between the threshold and the amount of qualified query agents.

Traffic cost is one of the most important parameters on which network administrators focus their concerns. Some network administrators would refuse P2P applications for heavy traffic.

Response time is the parameter which is of concern by P2P users. We define response time of a query as the time period from when the query is issued until when the source peer receives a response result from the first responder.

Security overhead is defined as the cost spent in the encryption and decryption of RSA and DES and Shamir's secret sharing scheme in SSMP.

5.2 Simulation Methodology

We simulate P2P topologies with DSS Clip2 trace [6]. The results are consistent with different traces and here we show two of them: Dec 28, 2001 and May 29, 2001, denoted as Trace 1 and Trace 2, respectively.

In our simulation, we implement flooding search used in a decentralized P2P network by conducting Breadth First Search algorithm from a specific node. A search operation is simulated by randomly choosing a peer as the sender, and a keyword according to Zipf [9] distribution. In each run, 1,000 search operations are simulated.

We also run the crypto software kits on a desktop PC with PIII 800MHz CPU, 256MBytes memory, 20G harddisk, and 10/100M Ethernet card and observe some hardware crypto servers performance. We found that the average 1024-bit RSA decryption rates are from 14 to 103.09 per second. The encryption rates are from 275 to 1941.7 per second. Normally the RSA encryption is 10~19 times faster than the decryption process. The 768-bit and 512-bit RSA test results also validate this. In most cases, the 1024-bit RSA can be regarded as an enough security cryptography algorithm. In our simulation we choose the 1024-bit RSA as the crypto process in the onion path and use 45.87 and 864 per second, which are close to the average level of our samples, as the reference value of the 1024-bit RSA performance.

Meanwhile, our statistics show that DES performance is in the range from 2.24Mbps to 8.78Mbps, and 3DES is from 1.89Mbps to 6.43Mbps. The hash function of MD5 performance is from 0.97Mbps to 11.64Mbps, and SHA-1 is from 3.23Mbps to 11.28Mbps. Therefore, we choose 5.41Mbps as DES speed and 7Mbps as SHA-1 speed.

In P2P networks, peers join and leave frequently. We simulate the dynamic peer changes by assigning a lifetime

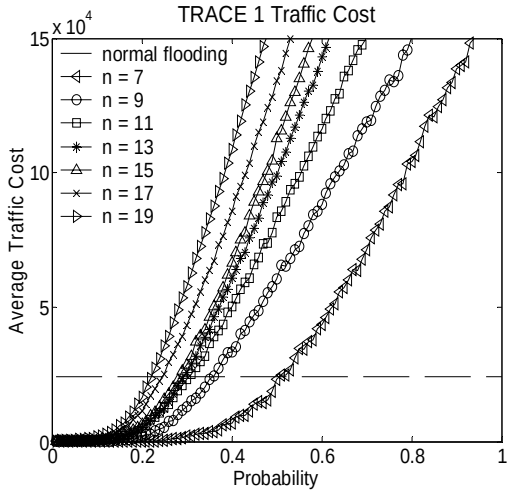

Figure 5: Traffic cost-Trace1

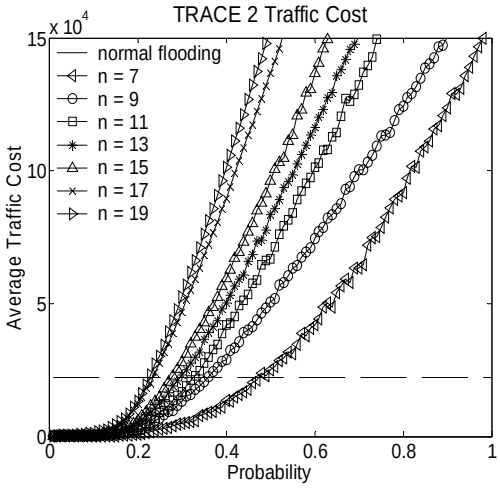

Figure 6: Traffic cost-Trace2

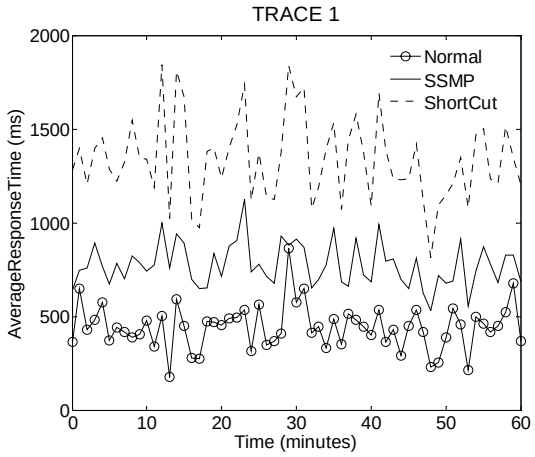

Figure 7: Response time-Trace 1

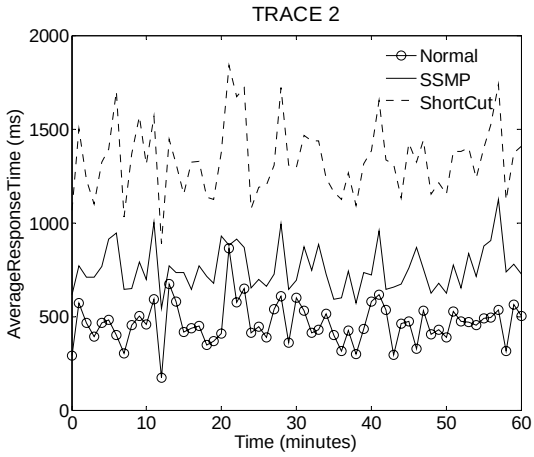

Figure 8: Response time-Trace 2

in seconds to every peer. The average of this value is 10 minutes. The lifetime decreases by one after each passing second. When a peer's lifetime reaches zero, it leaves in the next second. After a certain number of peers leave the network, we then randomly pick up the same number of peers from the physical network to join the P2P overlay.

5.3 Performance Evaluation

The Query scope in our protocol is relevant to the average flooding probability and the threshold of Shamir's scheme. Each peer produces a different share distribution in different average flooding probabilities. We show typical distributions of the shares from the two traces. We choose two nodes whose IDs are 1189 in Trace 1 and 1125 in Trace 2. They have 11 neighbors in the P2P topology. Figures 3 and 4 compare their shares distributions changed by different probabilities. The results show that the optimized threshold is very close to $\lfloor (1-p_v)\times d \rfloor$, where p_v is the average probability and d is the number of share receiving peers selected by the initiator. SSMP allows peers to flexibly choose the threshold to optimize the balance between the number of the query agents and the traffic cost.

The traffic cost added by SSMP is mainly caused by share flooding. We show the average query cost of Trace 1 and Trace 2 in Figures 5 and 6, respectively. Indeed, the more the shares are split, the higher the traffic cost is.

In Figure 5, when the split share number is 11 and probability is 0.3, the average increased traffic cost is 23097, which is close to 24189, an average normal flooding traffic cost in Trace1. If the average probability defined in SSMP increases, the traffic cost grows as well. In this design, we set the default average probability as 0.3.

Figures 7 and 8 show that the additional time caused by SSMP is about 20-40% more than that of normal query response time, while Shortcut-responding Protocol is 40-60% longer. We also notice that reducing the

Table 2: Major security processes with different algorithms in SSMP

Shamir Secret sharing, IDA		*SIDA* Scheme		RSA		DES
Distribution	Reconstruction	Distribution	Reconstruction	Encryption	Decryption	Encryption, Decryption
1, 1	l, l	1	1	$2t$+1	$2t$+1	2, 2

threshold of Shamir's scheme may offer a fast result-return by having more middle agents. However, the traffic would increase.

There are two main reasons for Shortcut-responding Protocol to take more time than that of SSMP. First, Shortcut-responding Protocol uses the probability forwarding method in the very beginning of the query spreading procedure. Although finally a query will flood after it reaches a certain peer that devotes itself as the reply node, the former path on which the queries have transferred is not the shortest path between the initiator and the reply node. Second, the delay time may be reduced in SSMP if the agent peers are placed exactly in the shortest path between initiator and responder. SSMP also obtains more than one agent peer, so the distribution probability of agent peers being on the shortest path between the initiator and responder is higher than that of Shortcut-responding Protocol.

The security overhead includes two parts, the processing overhead of the cryptography and the additional space requirement by the cryptography algorithms. Main security process includes share distribution and reconstruction procedure, RSA encryption and decryption, and DES encryption and decryption. If the average number of agent peers is l and the average length of Onion path is t, we show the number of security related operations among relevant peers in a complete search procedure in Table 2. We can see that the numbers of security related operations are very low.

6. Conclusion

In this paper, we propose a mutual anonymity protocol, called Secret-sharing-based Mutual Anonymity Protocol (SSMP), in decentralized P2P systems. SSMP employs Shamirs' secret sharing scheme to let peers issue queries and let responders deliver requested files anonymously. Compared with existing designs, SSMP achieves mutual anonymity in P2P systems with a higher degree of anonymity and a lower cryptographic processing overhead. We evaluate SSMP by comprehensive trace-driven simulations.

Future work on SSMP will lead in two directions. One is to improve the query flooding model of decentralized P2P systems by using other secret sharing schemes to acquire efficiency and introducing advanced protocols such as Tor [15] to improve security. The other one is to deploy the SSMP prototype combined with the construction of Freenet system to achieve the publisher anonymity in P2P systems. Note that the publisher anonymity in decentralized P2P systems can also be improved by employing SSMP protocol.

7. References

[1] KaZaA, http://www.kazaa.com

[2] Gnutella, http://gnutella.wego.com/

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

[4] Anonymity, http://freehaven.net/anonbib/topic.html

[5] Gnutella Protocol Development, http://rfc-gnutella.sourceforge.net/index.html

[6] The Gnutella Protocol Specification v4.0, http ://www.clip2.com/GnutellaProtocol04.pdf

[7] A. Back, I. Goldberg, and A. Shostack, "Freedom Systems 2.1 Security Issues and Analysis," *Zero Knowledge Systems, Inc. White Paper*, 2001.

[8] K. Bennett and C. Grothoff, "GAP - Practical anonymous networking," in Proceedings of Privacy Enhancing Technologies workshop, 2003.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in Proceedings of IEEE INFOCOM, 1999.

[10] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," in Proceedings of ACM SIGCOMM, 2003.

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in Proceedings of Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, 2000.

[12] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-peer Networks," in Proceedings of ACM SIGCOMM, 2002.

[13] R. Dingledine, M. J. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," in Proceedings of Workshop on Design Issues in Anonymity and Unobservability, 2000.

[14] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in Proceedings of the 13th USENIX Security Symposium, 2004.

[15] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in Proceedings of the 13th USENIX Security Symposium, 2004.

[16] M. Freedman and R. Morris, "Tarzan: A Peer-to-Peer Anonymizing Network Layer," in Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS), 2002.

[17] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, 1999.

[18] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," in

Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), 2003.

[19] W. Jia, D. Xuan, W. Tu, L. Lin, and W. Zhao, "Distributed Admission Control for Anycast Flows," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2004.

[20] S. Jiang, L. Guo, and X. Zhang, "LightFlood: An Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems," in Proceedings of International Conference on Parallel Processing (ICPP), 2003.

[21] H. Krawczyk, "Distributed Fingerprints and Secure Information Dispersal," in Proceedings of the 12th annual ACM symposium on Principles of distributed computing, 1993.

[22] H. Krawczyk, "Secret sharing made short," in Proceedings of the 13th annual of International cryptology conference on Advances in cryptology, 1994.

[23] D. Kugler, "An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks," in Proceedings of Privacy Enhancing Technologies workshop, 2003.

[24] K. Kurosawa and T. Takagi, "Some RSA-Based Encryption Schemes with Tight Security Reduction," *ASIACRYPT*, 2003.

[25] Y. Liu, X. Liu, L. Xiao, L. M. Ni, and X. Zhang, "Location-Aware Topology Matching in Unstructured P2P Systems," in Proceedings of IEEE INFOCOM, 2004.

[26] Y. Liu, L. Xiao, and L. M. Ni, "Building a Scalable Bipartite P2P Overlay Network," in Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS), 2004.

[27] Y. Liu, Z. Zhuang, L. Xiao, and L. M. Ni, "A Distributed Approach to Solving Overlay Mismatch Problem," in Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), 2004.

[28] Y. Liu, L. Xiao, X. Liu, L. M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2005.

[29] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein, "Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers," in Proceedings of ACM International Conference on Computer and Communications Security (CCS), 2003.

[30] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *ACM JACM*, 1989.

[31] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web Transactions," *ACM Transactions on Information and System Security*, 1998.

[32] Rennhard and B. Plattner, "Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection," in Proceedings of ACM workshop on Privacy in the Electronic Society, 2002.

[33] V. Scarlata, B. N. Levine, and C. Shields, "Responder Anonymity and Anonymous Peer-to-Peer File Sharing," in Proceedings of the 9th International Conference of Network Protocol (ICNP), 2001.

[34] A. Serjantov, "Anonymizing Censorship Resistant Systems," in Proceedings of the First International Workshop on Peer-to-peer Systems, 2002.

[35] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[36] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A Protocol for Scalable Anonymous Communication," in Proceedings of IEEE Symposium on Security and Privacy, 2002.

[37] C. Shields and B. N. Levine, "A Protocol for Anonymous Communication over the Internet," in Proceedings of the 7th ACM Conference on Computer and Communication Security (ACM CCS), 2000.

[38] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," in Proceedings of IEEE INFOCOM, 2003.

[39] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous Connections and Onion Routing," in Proceedings of IEEE Symposium on Security and Privacy, 1997.

[40] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: A Robust, Tamper-evident, Censorship-resistant Web Publishing System," in Proceedings of the 9th USENIX Security Symposium, 2000.

[41] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed Caching and Adaptive Search in Multilayer P2P Networks," in Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), 2004.

[42] M. Wright, M. Adler, B. N. Levine, and C. Shields, "An analysis of the degradation of anonymous protocols.," in Proceedings of the 9th annual of Symposium of Network and Distributed System Security, 2002.

[43] L. Xiao, Z. Xu, and X. Zhang, "Low-cost and Reliable Mutual Anonymity Protocols in Peer-to-Peer Networks," *IEEE Transactions on Parallel and Distributed Systems*, 2003.

[44] A. Yaar, A. Perrig, and D. Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," in Proceedings of IEEE Symposium on Security and Privacy, 2003.