

# Censorship-Resistant and Anonymous P2P Filesharing

Regine Endsuleit and Thilo Mie

Universität Karlsruhe (TH), Germany  
E-mail: {endsuleit, mie}@ira.uka.de

## Abstract

*We present a P2P filesharing system that allows redundant storage of shared files in a way that no participating server ever stores data that could compromise its operator. Instead, only fragments that do not contain any information about the original file in the information theoretic sense are uploaded. Reconstruction of a file requires all fragments it has been decomposed into. By this, in conjunction with other cryptographic methods, we yield significant legal advantages for server operators, as well as censorship-resistance, anonymity, secure routing, authenticated file update and integrity checks.*

**Key Words:** Peer-to-Peer Systems, Anonymity, Censorship Resistance, Encryption, Structured Overlay Networks

## 1 Introduction

P2P filesharing is a new application for the Internet and has become very popular recently. The technology used in these systems is still evolving, but already usable on a large scale. While not yet as important as email or the WWW, the number of concurrent users is already in the millions [11, 5, 4]. The legal implications of the availability of these systems are unclear at this time. While many users of P2P filesharing frequently violate copyright, the overall user population seems to be large enough to actually change copyright itself in order to accommodate new forms of media distribution and usage. The situation is made more difficult because newer P2P filesharing systems may contain anti-censorship technology, which on the one hand promotes free speech in countries where free speech is not a legal right. On the other hand these technologies obscure how copyrighted works are handled in P2P filesharing systems.

In all currently used filesharing systems, the data being shared resides directly on the hosts. If these computers are seized, their owners can be held accountable.

To prevent from this, Freenet [2] goes a step farther by encrypting the shared files.

From a legal and information theoretic point of view an encrypted file in this system still contains the whole information about the original data, although it is unreadable. In the future this could be prohibited by law and thus the server operator could be held to account for the contents, anyhow. With current law, this can happen when no secure key management is available and the encryption is broken.

In this paper we present a new approach that offers a strong protection for all clients and storage nodes from legal prosecution and censorship. Thus, it strongly supports free speech. We build our system on two stones: First, we introduce a strong authentication of all servants that on the one hand allows to use blacklists for servants that behave somehow illegal or un-ethical (e.g. by trying to spam the system) as well as authenticated file update and on the other hand when registering at the certification authority a servant operator gets a pseudonym. This is done in a way that neither the pseudonym itself reveals any information about the true identity nor can the certification authority be (illegally or legally) be forced to publish the real-life identity for a given pseudonym. Thus, although signing those files a server operator wishes to share with other people, he cannot be held to account for their contents.

On the other hand we prevent server operators from being held to account for the contents of files they allow to store on their server. We use a symmetric cypher to encrypt the original file before generating two fragments by means of a one-time pad. The latter guarantees perfect security in an information theoretic sense as both fragments must be combined to yield any information about the encrypted file. At no point of time any party participating in storing or routing the fragments ever owns the least information about the original file. Only the Internet service provider (ISP) of a client could (be forced to) collect the single fragments and reconstruct the data. This is a difficult task because one ISP is responsible for thousands of clients

and thus has to store and process a lot of traffic. But, one cannot ignore the risk that a few specific clients could be kept under surveillance. Therefore, the information is additionally encrypted and thus unreadable. The encryption is not very hard (similar to Freenet) but it makes attacks more difficult and most probably only clients that behave illegal will be victims of this kind of attack (see also [7]).

Our approach proves that it is possible to create a P2P filesharing system that cannot be spied out by server operators. During the whole upload and download process it is not possible to gain any information about the contents that is processed. The system also provides secure routing, redundancy and, thus, robustness (see e.g. [9]).

The only way to prohibit such a system seems to make P2P filesharing systems outright illegal or impose strong limitations. It seems doubtful that the implied negative impact on personal freedom is justified.

In our opinion it is worth it to look at P2P systems that offer strong security guarantees although they might be less efficient. Besides possible censorship, enterprises could have an interest in storing sensitive data distributedly and encrypted and thus in a redundant and secured way. There could be national laws enforcing data to be only readable and processed within one country. Although from cryptographic point of view there is no relevant difference between storing encrypted files on one computer or distributing encrypted files, but from an economic and juridical point of view there are advantages:

1. Redundant distributed storage allows the whole system to degrade gracefully since a predefined minimum number of servers must be corrupted until the information stored gets lost. In addition, the efforts an attacker must undertake to break the whole system increases with the number of distributed servers.
2. Current law in the United States forced Napster's central server to shut down. Storing whole (and partly copyrighted) files is the main reason for this. Fragmentation of the data may prevent from this.

The remainder of this paper is structured as follows. In section 2 we will present our P2P filesharing system by starting with a discussion about our security mechanisms. We then explain the routing and storing of files that are to be shared or downloaded. Section 3 summarizes the the security properties we have achieved. Finally, in section 4 we compare our system with other systems like Freenet offering some security. We will conclude in section 5 with a discussion about what has been done and what is needed for future work.

## 2 System Outline

Today, most P2P systems in use assume the data that is to be shared unencrypted. That causes the problem of possible censorship and holding hosts operators and users responsible for the contents of data they store and download, respectively. Actually, international law requires host operators to inform the official public authority in case they have any suspect that data could be copyrighted or contain illegal contents. A first solution to this is the P2P system *Freenet* in which all data stored in the network is encrypted. But in fact, files are still stored as a whole on each host. The decryption key is gained by searching for the data with the right keywords. Thus, finding the right keywords needs to be simple for the peers and the clients, respectively. Future laws may be as restrictive to prosecute even holders of encrypted data if there is a reasonable chance of decryption. Thus, most existing peer-to-peer systems cannot prevent people governed by dictatorial regimes from being prosecuted for publishing their (probably inconvenient) opinions.

Our approach goes farther and consists of the following components:

1. Strong user authentication via a CA. Each participant gets two certificates, One that contains a pseudo identity (pseudoID)  $P$ , the other one a random node identification number (nodeID). The first serves to sign documents in order to allow authenticated update and integrity checks, the other guarantees a random distribution of node identifiers and secure routing (virtual private network). Both certificates are blindly signed so that the CA can neither reveal the true identity or a connection between pseudo identities and nodeIDs later-on.
2. Generation of two fragments out of the encrypted file in a way that a single fragment contains no information (in the information theoretic sense) and reconstruction is possible if and only if both fragments are available.
3. Encryption of the original file via symmetric encryption like AES to prevent ISPs from reading downloaded files.
4. Cryptographic hash values generated from keywords which describe the file are used to generate an encryption key and two file identifiers  $id_i$ , ( $i = 1, 2$ ).

When downloading a fragment we do not want anyone but the client and storage node to be aware of the fragment's identifier. This design choice rules out store-forward networks like Freenet [2]. Because the nodes along up- and download routes see the fragment's id and decide whether storing or just forwarding the message. In fact, it requires a structured P2P network or more precisely a locally computable map from the fragment id to the id of the responsible node. Hence, for our purpose structured P2P overlay networks like Chord [12], CAN [8] and Pastry [10] are of great interest, because they implement such a map as distributed hash tables. Our storage system lies on top of such a structured P2P network and obviously inherits the security strengths and weaknesses, respectively, of the chosen P2P network. We have selected Pastry as underlying network with its secure routing primitive introduced by Castro et. al. [1]. Because it offers fast locality aware routing if no faults occur and gracefully degrades to a secure but slower routing in the presence of malicious behavior.

Using the secure routing mechanism of the Pastry network we have to demand the users to register at a certification authority. We wish to have unique identities within the system to enable authorized file update, black lists and signed fragments. On the other hand one of our goals is the protection of each and every participating party against legal threats such as for example censorship. Thus, the CA has to certify pseudonyms instead of real identities. Since the CA itself could be attacked and reveal the real identity belonging to a pseudonym we use a cryptographic method from the domain of digital money where anonymity of the clients plays an important role, too. The registration process is as follows:

- In order to participate in the system, a user  $A$  has to prove his identity to the CA as usual (passport etc.).
- $A$  generates a pseudoID  $P$  together with a public/private key pair  $(e_P, d_P)$  and a random value  $r_1$ .  $A$  sends  $c := r^{e_{CA}}$  where  $e_{CA}$  is the CA's public key. The CA returns  $a := c^{d_{CA}}$  to  $A$  who then computes his pseudoID as  $a_{CA}^e/r$ . The private key is used to authenticate a specific  $P$  as file owner.
- In order to assign  $A$  a unique nodeID together with a public/private key pair  $(e_S, d_S)$  (for general authentication purposes) that does not allow any connection to pseudoID  $P$ ,  $A$  generates random numbers  $r_1, r_2, r_3$  and a public key  $e_S$  and

sends  $(c_1, c_2) := (r_2^{e_{CA}} \cdot r_1 \cdot e_S, e \cdot r_3^{e_{CA}})$  to the CA. The CA computes a random value  $r_4$  and returns  $a_1 := (c_1 \cdot r_4)^{d_{CA}}$  and  $a_2 := c_2^{d_{CA}}$  to  $A$ . The public key  $e_S$  is then  $(a_2/r_2)^{e_{CA}}$  and the nodeID is defined as  $S := c_1/(r_2 \cdot e_S^{d_{CA}})^{e_{CA}}$ .

Proceeding like this, the CA can ensure that a single participant cannot register with different names (spoofing). At the same time, an attacker cannot force the CA to reveal an identity for a given pseudonym, since the CA does not know it. Furthermore, there is no connection between nodeID and pseudoID. Thus, no-one can determine the nodeID from which files that have been signed with the pseudoID  $P$  have been uploaded.

In this section we will present the way we pre-process the files that are to be shared. In fact, the level of security is directly dependent on the quality of the random numbers used. For perfect security in the information-theoretic sense true randomness is needed, making the approach at least problematic in practice. However there is a trend to include hardware random number generators into new systems, which might make this approach reasonably efficient in the future.

The file fragmentation process requires the use of a cryptographic hash function  $h$  that generates 160 bit hash values. *Fragment generation:*

1. User  $U$  fixes some keywords  $k_1, \dots, k_l$  that describe a file  $m$ . They will be used for the search mechanism. To prevent the hosts from retrieving information about the contents of the original file, the concatenated and normalized keywords are transformed into a hash value  $H := h(k_1 | \dots | k_l)$ .
2.  $U$  generates a key  $k$  for a symmetric encryption by computing  $k$  as the 128 most significant bits of  $H$ .
3. Two *global* hash values  $id_1 := h(H|1)$  and  $id_2 := h(H|2)$  are generated via concatenation of the keywords hash and a fragment index. Both are used as identifiers for the two packages.
4. The message  $m$  is encrypted to  $enc((m), k)$  via e.g. AES with  $k$  as key.
5. User with pseudoID  $p$  computes one random number  $r$  of length  $m$  and generates the fragments

$$\begin{aligned} p_1 &:= sig((r \oplus E_k(m), E_k(id_1), e_P), d_P), \\ p_2 &:= sig((r, E_k(id_2), e_P), d_P) \end{aligned}$$

The fragments are signed with the private key  $d_p$  that has been certified together with the pseudoID  $p$ .

We use a cryptographic hash function  $h$  to compute the keyword hashes. The use of a *cryptographic* hash function in step 1 is inevitable in order to generate strong keys, and in step 3 our storage method requires two sufficiently different search indices for the fragments to ensure that on each storage node at most one fragment is stored. This is guaranteed by a cryptographic hash function.

In this section we present how the fragments of a file are replicated and stored in a Pastry network at locations associated with their 160 bit identifiers  $id_1$  and  $id_2$ , respectively. At first we briefly introduce the Pastry architecture with particular stress on security issues to keep our paper self contained. For more details on Pastry we refer to the original papers [1, 10].

Each node in a Pastry network has a randomly chosen nodeID  $S$  out of a circular 128 bit address space and maintains a small routing table as well as a set  $\mathcal{N}$  of online neighbors. The routing table holds up to 128 IP addresses of other nodes and is organized in  $128/2^b$  rows and  $2^b$  columns. Usually a Pastry node chooses the entries in its routing table by network proximity and nodeIDs to improve routing performance. Unfortunately, the adversary could exploit the network proximity and make most entries in honest nodes' routing tables point to malicious nodes. Castro et. al. have proposed in [1] a hybrid routing approach that routes aware of network proximity, tests on the presence of faults and resends messages by secure but slower routing if faults are detected. We do not describe this approach here and limit our presentation to the secure and not network proximity aware routing. For addressing the routing table entries the 128 bit addresses are split into digits of  $b$  bits each. The table's entry  $(i, j)$  is filled with the IP address of a node whose nodeID shares all but the  $(i + 1)$ th  $b$ -ary digit with the current nodes nodeID and the  $(i + 1)$ th  $b$ -ary digit is equal to  $j$ . A routing table entry is left empty if no node with appropriate nodeID prefix is known or its nodeID prefix equals to the  $i + 1$  MDs of the current node. The parameter  $b$  affects the number of required hops needed to deliver a message as well as the size of  $\mathcal{N}$  and is typically set to 4. A nodes  $N$  neighbor set  $\mathcal{N}$  contains the online nodes with the  $|\mathcal{N}|$  nodeIDs which are numerically closest to the nodeID  $S$  of  $N$  (including  $S$ ). The set's size  $|\mathcal{N}|$  is typically chosen to be  $\lceil 8 * \log_{2^b} n \rceil$  with  $n$  the number of expected nodes in the network. Each node keeps track of its neighbor set and reacts on failing and joining nodes appropriately. The neighbor set ensures reliable message delivery and fault tolerance by

storing replicas of the file fragments as we will describe now.

When storing or retrieving a fragment with the 160 bit identifier  $id_i$  we need to find the online node whose nodeID  $S$  is numerically closest to the 128 most significant bits (MSBs), called  $id'_i$ , of  $id_i$ . The fragment will be stored on this node and the fragment's replicas on the  $c \leq |\mathcal{N}|$  neighbors whose nodeIDs are numerically closest to  $S$ . The nodes make sure that this invariant is always valid by copying the fragments to all nodes entering the network whose nodeID is numerically closer than the old ones. At the same time, nodes that do not belong to the  $c$  closest ones anymore delete their copy of the fragments. The number  $c$  of replica should reflect the rate of likely faults and transient nodes. During upload a client finds the node with nodeID  $S$  by sending a message to a nodeID whose 128 MSBs  $id'$  are equal to  $id'_i$ . For matters of simplicity let us first assume all nodes to be honest. In this case the message is delivered to the online node who's nodeID is numerically closest to  $id'_i$  after maximal  $\lceil \log_{2^b} n \rceil$  hops. This is because each node  $N$  forwards incoming messages to a node in its routing table, that shares at least one more leading  $b$ -ary digit with  $id'_i$  than  $id'_i$  is sharing with  $N$ 's nodeID. Should no appropriate node be found in either the routing table or the node's neighbor set, then the current node or its neighbor that has a nodeID that is next in size to  $N$ 's nodeID is the message's final destination. This is correct, since  $id'_i$  is either equal to the current nodeID  $id'$  or lies numerically between the current and the numerically succeeding online node's id. Now, fragments can be stored or retrieved, resp., by routing messages to the fragment  $id'_i$ . As the Pastry nodes always keep track of their neighbor set and keep the number of replicas constant if not all  $c$  copies of a fragment get suddenly lost by simultaneous node failures within one recovery period.

However, in contrast to the earlier, simplifying assumption we cannot assume all the nodes of a network to be honest. Much more to the contrary, we have to assume that there is a fraction  $f$  of malicious nodes that conspires. In this setting routing has to be secure enough that a message for an identifier  $id_i$  is eventually delivered to all legitimate online nodes holding replicas for this identifier. Despite, the adversary might drop, misroute, corrupt the message or even impersonate legitimate nodes. According to Castro et. al. [1] the implementation of such a secure routing primitive is depending on three key ingredients:

- Routing tables have to be maintained securely. As we already remarked earlier, there have to be strong constraints on the appropriate table entries such that routing tables cannot be exploited by

adversaries.

- Node identifiers have to be assigned securely, randomly and uniformly distributed. If malicious nodes could choose their id, they could get full control over all replicas of one fragment.
- Messages have to be forwarded securely as even a small fraction  $f$  of malicious nodes can decrease the probability of successful delivery by simply not forwarding messages at all.

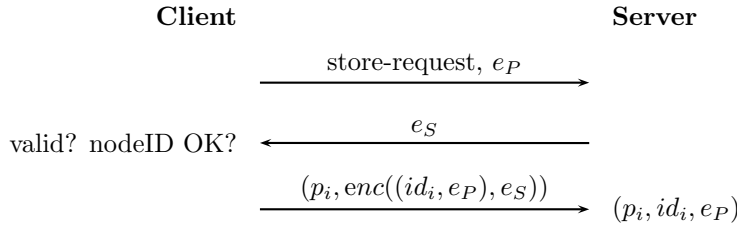
The first ingredient has been implicitly realized as we already defined our routing tables to be strongly constrained. In order to implement the second one, Castro et. al [1] propose to set up a Certificate Authority that assigns to each node a random nodeID and signs it together with the node's public key. As explained in section 2.2, we have modified this approach by using blindly signed nodeIDs. Our approach can even prevent adversaries from simulating thousands of malicious nodes on a single computer and thereby pushing the fraction of malicious nodes beyond any tolerable threshold (Sybil-attack). However, this approach is perfect for corporate environments, it is less desirable for open communities like the Internet. But unfortunately there is no other practical measure against Sybil-Attacks. Douceur even shows in [3] that this type of attack is always possible without a logically centralized authority or without meeting extreme and unrealistic assumptions on the nodes global coordination. Finally, the third ingredient is realized by sending copies of a message for a 128 bit nodeID  $S$  over multiple routes to each node holding a replica for  $S$ . How the aforementioned techniques are implemented in detail is described in [1]. Additionally, they show how to combine more efficient network proximity aware routing and secure routing.

Now, we have everything needed for a full description of the storing process of fragments with 160 bit identifier  $id_i$  in the Pastry network. As before, the 128 most significant bits of  $id_i$  shall be denoted as  $id'_i$ . The  $c$  online nodes with nodeIDs which are the numerically closest to  $id'_i$  are responsible for storing replicas of the fragment. Therefore, the client who wants to store the fragment in the network sends a store-request message to the key  $id'_i$ . The secure routing primitive ensures that all nodes responsible for storing the replica will receive this message with high probability. This message contains the client's public-key certificate and a request for the receiver to send his certificate. At least the honest nodes comply with this request. The client checks the certificates' validity and whether the senders' nodeIDs are those of the responsible replica nodes.

The client now sends to all verified replica nodes the package  $(p_i, enc((id_i, e_P), e_S))$ . The first component of the tuple,  $p_i$ , is one of the data packages from section 2.3, the second component consists of an RSA encryption of the full identifier  $id$  and the clients certified public key. The replica nodes decrypt the package and store it together with the fragment itself on the hard disk. Note, that this way only the client and the responsible replica nodes are aware of the fragment's full 160 bit identifier  $id_i$  and its source. Figure 1 shows the data flow between client and server when uploading file fragments.

When a user wants to retrieve a file from the network he first has to enter some appropriate keywords describing the contents. These keywords are normalised and hashed by his client. After generation of the two search indices  $id_1, id_2$  for the fragments, two messages are sent to addresses that are numerically close to the 128 MSBs of the identifiers  $id_1$  and  $id_2$ , respectively. The receivers reply to these messages by sending their public key certificates. Then, the client encrypts  $(id_i, e_P), i \in \{1, 2\}$ , with the public key of the particular receivers and sends this message to the receivers. The storage nodes send the signed fragments to the client who decrypts them and checks their integrity.

If a client wants to update a file it has stored in the P2P network, it splits the new version into fragments as he did with the old one. Note that the fragment identifiers are just dependent on the keywords which, from the point of view of the system, cannot change without generating new file fragments. But, one can request to delete the old version, so no disk space is wasted. Then, the client sends a securely routed update-request message to the key  $id'$  that equals to the 128 MSBs of the 160 bit fragment id. The message contains the client's public key  $d_P$ . At least the honest replica nodes of the fragment respond to the update-request message and send their public key certificates and a random number  $r'$  which is encrypted with the client's public key (to prevent from replay attacks) to the client. The client verifies the senders' certificates and checks whether they are really responsible replica nodes. Then, he decrypts the random number  $r'$  and encrypts the by one increased random number with his private key  $d_P$  and sends it to the sender of  $r'$ . In the end, it securely routes the new fragment to  $id'_i$ . Deleting goes along the same line but after sending  $r + 1$  encrypted to the storage node, no new fragment is to be transmitted and the old ones are deleted.



### 3 Results

By combining the advantages of a structured overlay network with strong cryptographic methods our approach provides not only a secure system in the cryptographic sense but also in the legal one.

1. *Protection of the storage nodes against legal prosecution*

A storage request includes only a data fragment that cannot be differentiated from a random number. It is not allowed to store the second fragment, too. This is regulated in the routing protocol we have presented in section 2.4 Thus, it is most probable that one server never owns the whole information about one specific file, even if the encryption of the original file is broken.

2. *Protection of the clients against legal prosecution*

Law could enforce Internet providers to check the data traffic of some specific people. To protect clients in case of downloading a file, it must be made impossible for the provider to reconstruct the original file. It is physically unavoidable that the ISP could collect fragments. To prevent from a possible reconstruction the original file is encrypted. As mentioned before, this encryption is not very hard, but it makes attacks from an ISPs side more difficult.

3. *Protection of the file owner against legal prosecution*

By providing each participant with a pseudonym an uploaded file can only be assigned to this identity. There is no possibility to find the true publisher even if the CA is attacked.

4. *Hardening DoS attack*

The single file fragments are signed and the authentication mechanism does not allow to generate numerous identities on one computer. This implies that for example by use of a black list a Denial-of-Service attack against one specific server within

the upload process is difficult. Furthermore, the integrity of the fragments can be checked by each node.

5. *Acceptable trade-off between security and routing/searching efficiency*

We base our system on the P2P system proposed by Castro et al. [1] to get an upper limit of servers that must be visited during one search process. Let  $n$  be the number of participating nodes. Then, at most  $\lceil \log_2 n \rceil$  nodes have to be visited till the right file is found. In contrast to this, Freenet offers no upper limit for search. In case of 10,000 nodes and a failure rate of up to 25% a request visits not more than 4 nodes in a Pastry network. A similar scenario causes an *average* of approx. 17 hops using Freenet [2].

6. *Protection against censorship*

All files are stored in an unreadable fashion and only reconstructible by the user downloading a file. A censor is not able change or remove information that has not been uploaded by him. Searching for the right keywords as e.g. “democracy” may deliver demolitionist material but he can not determine the source (neither the true identity nor the nodeID that belongs to the pseudoID).

### 4 Anonymity and Encryption in P2P Filesharing

There are efforts underway to include strong encryption and reliable anonymity into P2P filesharing systems. One example is MUTE [6] which uses localized routing based on the behavior of ants to obscure source and destination of data and queries but offers no encryption. Consequently, the servants storing files are not safe against censorship and could be held to account for the contents they store.

In Freenet contents of a file stored on a host is obscured because it is encrypted with a key not known to the host, but it still completely resides on the host. If

the encryption is broken, the file can be recovered. If the encryption has certain weaknesses, which is most probable since there is no secure key management, it may be possible to recognize an encrypted and specially prepared file without actual decryption.

In our system the only way a file can reside completely on a host is in temporary storage as result of a search initiated on that host. All fragments of a file stored in the network are unreadable in the sense that the original file content is not only encrypted, but in fact not there. The individual fragments created with the use of a one-time pad do not carry any information about the original file, except for the meta-data.

One important difference to Freenet is that our scheme does not allow dictionary attacks. In Freenet, successfully guessing the search string for a file allows decryption without performing the search. Since people searching for specific files need to have a realistic chance to guess the right search string, so can an automated system working from a dictionary of phrases and terms. Our system does not prevent from ISPs performing a dictionary attack, but no-one else can gain information.

Furthermore, Freenet requires in average and theoretically more hops for retrieving files than a Pastry-based network like ours. This is a result of being an unstructured network. It additionally opens the possibility of various attacks against non-secured node assignments and malicious routing table updates.

Also, the combination of strong authentication and pseudonyms is a feature, Freenet does not have and thus it is not as resistant against various attacks as ours.

## 5 Conclusion and Future Work

We have described how an existing system can be modified to implement the idea of preventing censorship and legal prosecution of all participants. Also, it is possible to check the integrity of the downloaded files and to secure file updates by an authorization. For each file inserted into the network it requires truly random data once the size of the file to be inserted. While that limits applicability at the moment, future computers equipped with hardware random number generators may make this option efficient enough.

By using an encryption a caching based P2P file-sharing system can be made more resilient against legal attacks. One possible impact of our proposed modifications is not the technical improvement, but rather an improvement in the legal defenses host operators could use. Although it seems strange to do technical adjustments to satisfy purely legal constraints, this type of

effort becomes more and more common today.

We have started to analyze the presented approach as a basis for a P2P Internet search machine or even anonymous Internet applications. Using some data mining techniques the system gains scalability (in this paper the search space is restricted by the size of the hash values). Furthermore, we are using techniques from private information retrieval (PIR), session keys and mixers. The combination of these tools eliminates the remaining threats like DoS attacks on database index tables, traffic analysis performed by ISPs and anonymity between client and server.

## References

- [1] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 299–314, 2002.
- [2] I. Clarke, S. Miller, T. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.
- [3] J. Douceur. The sybil attack. In *Proc. of the IPTPS02 Workshop*, pages 251–260, 2002.
- [4] P. Jardas. P2P Filesharing Systems: Real World NetFlow Traffic Characterization. Master's thesis, ETH Zürich, February 2004. <ftp://www.tik.ee.ethz.ch/pub/students/2003-2004-Wi/BA-2004-01.pdf>.
- [5] T. Mennecke. Bittorrent statistics. <http://www.slyck.com/news.php?story=370>, January 2004.
- [6] <http://mute-net.sourceforge.net/index.shtml>.
- [7] Online copyright infringement liability limitation act. Electronically under: [http://en.wikipedia.org/wiki/Online\\_Copyright\\_Infringement\\_Liability\\_Limitation\\_Act](http://en.wikipedia.org/wiki/Online_Copyright_Infringement_Liability_Limitation_Act), 2006.
- [8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM'01*, 2001.
- [9] R. Rodrigues, B. Liskov, and L. Shrira. The design of a robust peer-to-peer system, 2002. 10th ACM SIGOPS European Workshop.
- [10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of FIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
- [11] Slyck. <http://www.slyck.com/>.
- [12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM'01*, pages 149–160, 2001.