The Free Haven Project

Distributed Anonymous Storage Service

Seminar: Peer-to-Peer Networks

Prof. Christian Schindelhauer
Tutor: Arne Vater

Computer Networks and Telematics
Department of Computer Science
Faculty of Applied Sciences
University of Freiburg

Author: Michael Janczyk

March, 2007

Contents

1 Introduction

At the time when the Free Haven Project was developed, the first lawsuits against peer-to-peer networks and their users were started. That saw to it that during this time many other peer-to-peer networks were started which wanted to raise the provided anonymity. But no other project went to such lengths. The Free Haven Project wanted to offer an anonymous, persistent and distributed storage to their users. In the end they failed due to their high requirements.

In the next sections the project will be scribed shortly, and the conclusion will mention the main reasons why the project has never been realized. At the beginning there is a brief discussion about anonymity: Why it's needed and what anonymity means in respect of the project?

After the short introduction on anonymity the design is specified: How to introduce servers to the service and how to remove them? How to insert and retrieve documents and how anonymity is achieved?

The last part is about the anonymous channel: Which goals were claimed and what problems occurred?

2 Anonymity

Many peer-to-peer projects use the word anonymity without any specification of its meaning. But without a closer look at the meaning, a distributed anonymous storage service cannot really be described. Even though the Free Haven Project annotated the lack of a formal definition of anonymity, they try to distinguish the responsibility of the service. For example: Is the system responsible for a user who signs a document with his *true name*? Is this system then still anonymous?

Instead of giving a formal definition the Free Haven Project tries to provide a basis for future analysis of this problem.

2.1 Why Is Anonymity Needed?

History shows that when data can be identified to be stored on a particular sever or can be allocated to a user (publisher, author or reader, → Figure 1), this user or server can become the victim of a prosecution. The prosecutors can be companies, parties, governments, or any other group or person, for whom that data could be classified to be disapproving or somehow negative. An example for that can be a political opponent who lives in a country where the government or the sovereign does not approve of any negative reports or of any opponents. But prosecution can happen to everybody, even in democratic countries. You will only need to share copyright protected material and the companies which claim the rights on that and even the law will try to hold you responsible for this.

But anonymity is a word which has to be examined in context because for each project anonymity can mean something else and full anonymity can be or maybe is impossible.

2.2 Partial Anonymity

As anonymity may be impossible the question which has to be raised is, *"Is the service anonymous enough?"*. One reason is that it is nearly impossible to hide all your traces. An example: For some reasons an adversary may be aware of some rough personal information

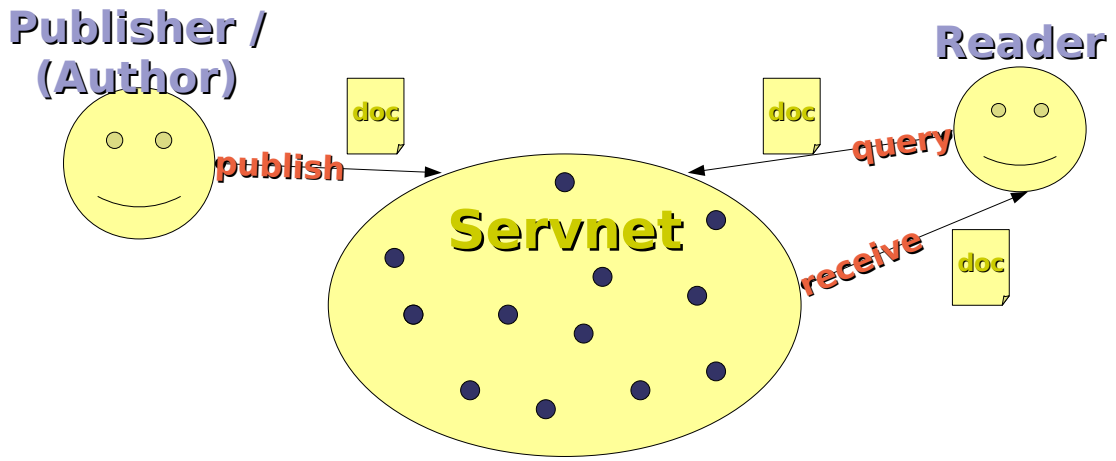

Figure 1: Entities and units of the Free Haven Project

of a user, like that he uses high-bandwidth connection and lives somewhere in California. Users which can be assigned into the same group form a *set of suspects*. This means that the service has to assure that this set is large enough, that it's not worth for an adversary to prosecute the whole set.

2.3 Pseudonymity

Pseudonymity is another important aspect. While anonymity does not allow any linking at all, pseudonymity describes two attributes of a transaction which can be linked. This is necessary for an anonymous system because it permits bidirectional communication, which is necessary to communicate with a server. Another aspect of pseudonyms is to allows the pseudonym to acquire reputation. This is very important in an anonymous system because it can limit server-caused damage by gaining reputation through good behavior. But both anonymity and pseudonymity protect the privacy of the *location* and the *true name* of a user or a server.

An example for a pseudonym could be:

> "[...], the documents digitally signed by 'Publius' could all be verified as 'belonging to Publius' without anyone coming to know who 'Publius' is in *'real life'*. [13]"

2.4 Anonymity For?

After given some basic thoughts about anonymity it can be defined what anonymity means for the entities, units and modules used by this project (→ Figure 1). Because communication already is discussed in depth [15] and it would go beyond the scope of this work, it will be left out in this discussion. What this work will only mention is that a strong anonymity of the communication channel is implicitly inevitable.

This work will only refer to the anonymity of the remaining entities and units. At first it is assumed hat an adversary picks an entity or a unit and tries to link it to another entity or unit. This leads to the following definitions:

- Author Anonymity:
  A system is author anonymous if an adversary cannot link a document to its author(s).
- Publisher Anonymity:
  A system is publisher anonymous if an adversary cannot link a document to its publisher.
- Reader Anonymity:
  A system is reader anonymous if an adversary cannot link a document to its readers.
- Server Anonymity:
  If an adversary knows a document's identifier, he is no closer to knowing which servers on the network currently possess this document.
- Document Anonymity:
  Document anonymity means that a server does not know which documents it is storing. There can be distinguished between two types of document anonymity:

  Passive-Server:
  This means that a server is only allowed to look at his own data. Anonymity can be easily be achieved by encrypting the document and storing the key on an external server. One possibility is to use **Secret Sharing** (→ Section 3.1.2) where the document is split up into n pieces where k pieces ($k \leq n$) suffice for the recreation of the document. That way the secret key is split up between the servers as well.

  Active-Server:
  In this case the server can participate as a reader in the network. It can communicate with other servers and compare its data to the data of other servers. This type of document anonymity is difficult to achieve, since it should be very difficult to differentiate between servers which act as readers and users who act as readers.
- Query Anonymity:
  Query anonymity means that the server cannot determine which document it is serving. A weaker form is *server deniability*. In this case the server knows the identity of the requested document, but no third party can be sure of the document's identity.

These specifications are crucial for a survival of an anonymous system, if breaking the anonymity of only one of the entities or units could imply danger for the entities involved.

Besides protecting the system from adversaries, anonymity provides *plausible deniability* [1] for the servers. Even if a server is taken by an adversary he cannot prove that this server has done something wrong.

3 Design

The Free Haven Project names many goals for the design specification. One of the most important is to use two independent modules, one for the publication system which will be

[1] "[...] little or no evidence of wrongdoing or abuse" [9]

discussed in this section, and one for the communications channel which will be external to the project. This allows modifications on each module without impacting functionality of the other. Thus giving the possibility to use an already existing communications channel, so that the project can concentrate on the publishing system which acts as a backend for that channel. The idea behind the Free Haven Project is a community of dedicated servers called the **Servnet**. This stands in contrast to the common thought of peer-to-peer networks where a server is a client as well.

To learn more about the communications channel please read section 4 on page 12.

3.1 Publication

Assuming a user ('Alice') wants to publish a document to the Free Haven Service, the following steps have to be done by the user / server. At the very beginning 'Alice' must find a server which is willing to store her document. Here she has the following possibilities:

- 'Alice' runs a server herself, therefore she has no problems in finding a server. On the other side she has much more administrative work to do because she has to configure and maintain the server.
- 'Alice' has to find a server which offers a public interface or a publicly available reply block (→ Section 4).

Once 'Alice' has found a server she can upload her document to that server. At this point 'Alice's' work is done and the server takes over the rest of the publishing. [2]

When the document F has arrived the server it is split up with the **Rabin's Information Dispersal Algorithm (IDA)** (→ Section 3.1.1) at first. **IDA** disperses the document into n shares $f_1, \ldots, f_n$, but any i shares f_i ($k \leq i \leq n$, where k indicates the minimum of shares needed) are sufficient to recreate the document.

After the shares are created the server computes a key-pair (PK_{doc}, SK_{doc}) for the document. The public key (PK_{doc}) is used as index for the document, the secret key (SK_{doc}) is needed for signing the document.

For every share this information is now stored in a data segment along with the share and is signed with the secret key. An example-share could look like this:

```
<share>
<PKdoc>cec41f889d75697304e89edbdddf243662d8c784</PKdoc>
<sharenum>1</sharenum>
<buddynum>0</buddynum>
<totalshares>100</totalshares>
<sufficientshares>60</sufficientshares>
<expiration>2000-06-11-22:25:24</expiration>
<data>Ascii-armored characters here</data>
<signature>cec41f889d75697304e89edbdddf243662d8c784</signature>
</share>
```

Where the public key is stored in `<PKdoc>` and the data of the document in `<data>`. All the information including the `<data>` block is signed with the secret key and the value is placed inside the `<signature>` tag. The other values will be explained later with the associated functions.

The share is now prepared and can be saved in the server's local space.

[2] Actually it can be considered to let 'Alice' do the next two steps, the splitting of the document and the creation of the key-pair, but it requires a lot more trust of 'Alice'

3.1.1 Creating Shares

The creation of shares is done with the **Rabin's Information Dispersal Algorithm** which will be described only sketchily here. For the whole information on **IDA** please read the corresponding paper [16].

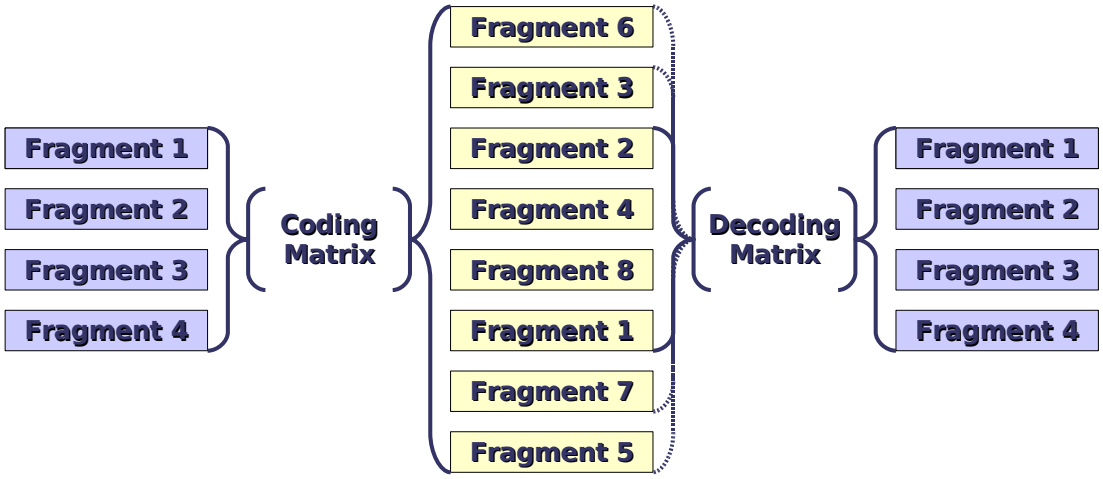

Figure 2: Rabin's Information Dispersal Algorithm
Source: Giampaolo Bella, Costantino Pistagna, Salvatore Riccobene. Distributed Backup through Information Dispersal. Università degli studi di Catania.
http://www-gris.det.uvigo.es/řebeca/vodca/slides/Bella_Pistagna_Riccobene_vodca04.pdf

The document is first split into k fragments. These fragments now are dispersed into n (`<totalshares>` in the below example) fragments ($n \geq k$) which represent our shares. These shares are now distributed between the servers in the servnet. If at some point a reader wants to rebuild the document he has to get any subset of i shares ($k \leq i \leq n$) from the servnet to recreate the document.

```
<share>
...
<totalshares>100</totalshares>
<sufficientshares>60</sufficientshares>
...
```

The robustness parameter k (`<sufficientshares>` in the above example) is based on the compromise between the importance of a document and its size. If a high k in relevance to n is used the document is brittle and could be unrecoverable after a few shares are lost. If you take $n = 100$ and $k = 99$ for example: This means there exist 100 shares and a minimum of 99 is needed to recreate the document. That document is unrecoverable after 2 shares are lost (→ Figure 3).

On the other hand, if you take a low value for k it indicates a large share, since there is stored much more data in each share. As an example you could take a movie: If you need only three shares to recreate the whole movie, each share has to be large.

This system provides anonymity for the document because you need at least k shares to recreate the document. As a side-effect even some redundancy ($r = n/k$) is gained.

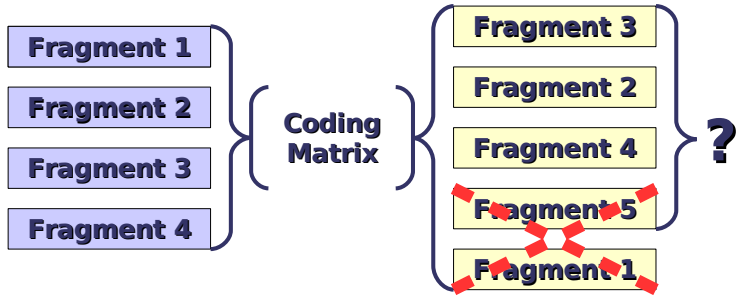

Figure 3: Unrecoverable document if too many shares are lost

3.1.2 Secret Sharing

The system which is used here is called **Secret Sharing**. First you have to specify in how many pieces you want to split your secret (k). To recreate this secret you will need at least k pieces. This can be easily shown with an example of **Blakely's scheme** [10]. To be able to show it graphically the example (Figures 4, 5 and 6) will use three dimensions which corresponds to $k = 3$, for more details see [16], [10].

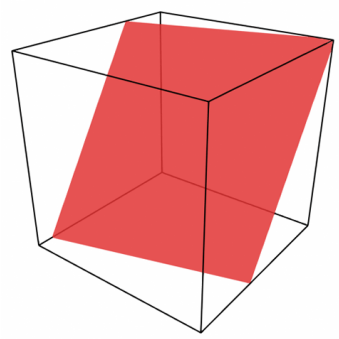

Figure 4: A plane symbolizes a share

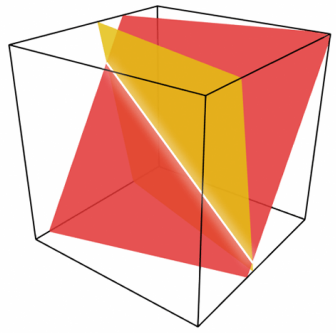

Figure 5: Two shares narrow down the secret to a line

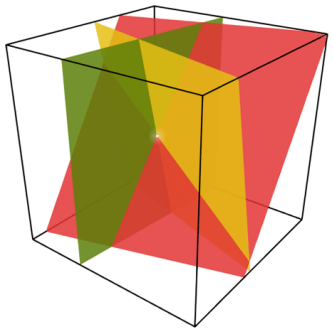

Figure 6: The intersection point represents the secret

Source: http://en.wikipedia.org/wiki/secret_sharing

In the example a plane symbolizes a share (Figure 4). With two shares you have not enough information to recreate the secret, but this will give you already enough information to narrow the secret down to a straight line (Figure 5). With three nonparallel plains you will get an intersection point which represents the secret (Figure 6). Once you've got the secret you can decrypt the document.

3.2 Retrieval

Users which want to retrieve documents from the Free Haven Project first have to find the corresponding index. This index would be the hash of the public key which was created during publishing. Searching for existing words like *"Metallica"* would suggest the content of this document and thus subvert anonymity. There will have to be websites or usenet groups where this hashes will link to their content, so that a reader can search for the document he wants. That will not be part of the Free Haven Service, so the user is reliant on himself to know where to find the correspondent hash.

Once this hash is obtained somehow the reader generates a key-pair $(PK_{client}, SK_{client})$ and an one-time remailer reply block. This public key will later be used for the encryption of the shares which will be send back to the reader. The way back to the user is an encrypted information inside the one-time remailer reply block, for more information about that way of communication please read [14], [15]. After all this is done the user has to submit this information to a server. This could happen through a secured web-interface where the trace to the user is hidden or an publicly available reply block.

When this information arrives the server which was chosen to do this request, that server broadcasts this request to the servnet including the document's index, the public key of the user and his one-time remailer reply block. Each server which retrieves this request checks its database if it contains this hash-value. When it is found the correspondent share is encrypted with the public key of the user (PK_{client}) and is sent back to the user using his reply block embedded in the broadcast.

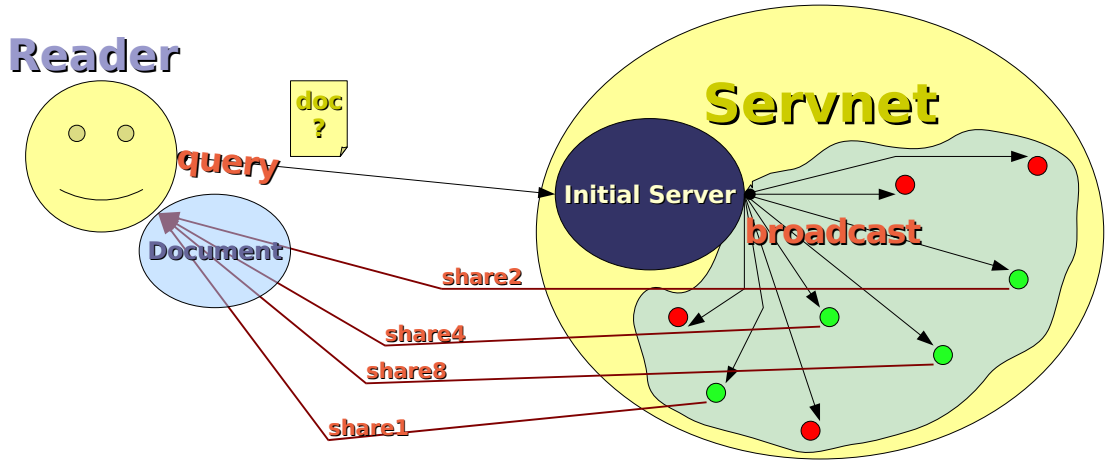

Figure 7: Document retrieval

For the reader it looks like the shares arrive out of the nowhere. Once the reader has received sufficient shares (k) he can recreate the document (→ Figure 7). A corresponding client should do this automatically. A problem could be that even k share suffice to recreate the document, up to n shares will be sent to the reader ($n \geq k$) because the system does not recognize when the reader has enough shares.

3.3 Expiration And Revocation

One of the key features of the Free Haven design is that they guarantee a publisher-specific lifetime for documents. It's an absolute time-stamp ('GMT') inside the share's data segment, e.g.: `<expiration>2000-06-11-22:25:24</expiration>`.

This means that no document can be removed from the system earlier without any consequences for the servers which deleted this particular document. But this means as well that some user (adversaries) can insert long-living documents no other user can use ("garbage") and which cannot be removed. Or maybe a document can get obsolete over the years or get some updates. A long lifetime would complicate the updating of this document since the old versions will still be found. Some other services like **Freenet** [14] or **Mojo Nation** [14] favor popular documents ('least recently used' (LRU)).

Revocation could be an answer to this question but it would involve new attacks as well and so complicate the design. That's why the Free Haven Project intended to let revocation out of the first implementation.

One way to achieve the possibility to revoke is to publish the hash of a private value into each share of a document. If the author at some point wishes to revoke this document he could send a revocation-message to the servnet including this private value. But as mentioned it would involve new problems, like a couple of new attacks on the system. Authors could use the same private value for revocation and so link their documents. Or an adversary who wants to damage an author's reputation could insert the same hash-value into dubious documents so that it would look like they are from the same author. This is possible without knowing the private value. The adversary only needs the hash-value and this can be obtained from a share. A revocation tag in a share would imply ownership to a document as well and so be an incentive for an adversary to find the user with the capability to revoke and force him to do so.

3.4 Trading

Trading is a function which enhances the anonymity of the service and makes it harder to attack special documents. Attacks on shares are more complicated when the share is moving from server to server rather than lying statically on a server. The anonymity of the publisher is also enhanced because trading provides cover for publishing. If trades are common nobody can be accused to be the publisher because this user could only be trading these shares. Another important aspect of trading is that shares with longer lifetimes are possible. If there is no possibility to trade shares, long-lived shares would be rather rare if it implied to keep these shares several years. It permits servers to leave the servnet properly without consequences for the service and the server's reputation. A server can trade his long-lived shares for short-lived ones if it wants to leave the network. That way no shares are lost.

The frequency of trades is set by the server itself, the server administrator respectively. If now a server ('Alice') wants to trade a share it first checks its database of known servers and chooses a server based on the reputation of this server (→ Section 3.6 and 3.7). If a server is chosen ('Bob'), 'Alice' sends the share to this server ('Bob') and asks for information about a return share. This information represents the *currency* of trading. It's a pair of variates '$size \times lifetime$' and should equal the value of 'Alice's' share. It is not clear what the operator '×' represents, but it can be presumed that as fixed-disk storage gets cheaper every day lifetime should weigh a lot more than size. It can be considered

that trades should be 'fair' or else a trade should not happen. One exception could be if a server is new and has no reputation and is forced to do some 'one-way' trades to gain reputation.

The protocol is a 4-round handshake (→ Figure 8). During the first two steps the shares are exchanged. In the third and fourth rounds receipts (→ Section 3.5) are being exchanged. These receipts give information about the traded shares and can be used as indicator if a trade has been successful.

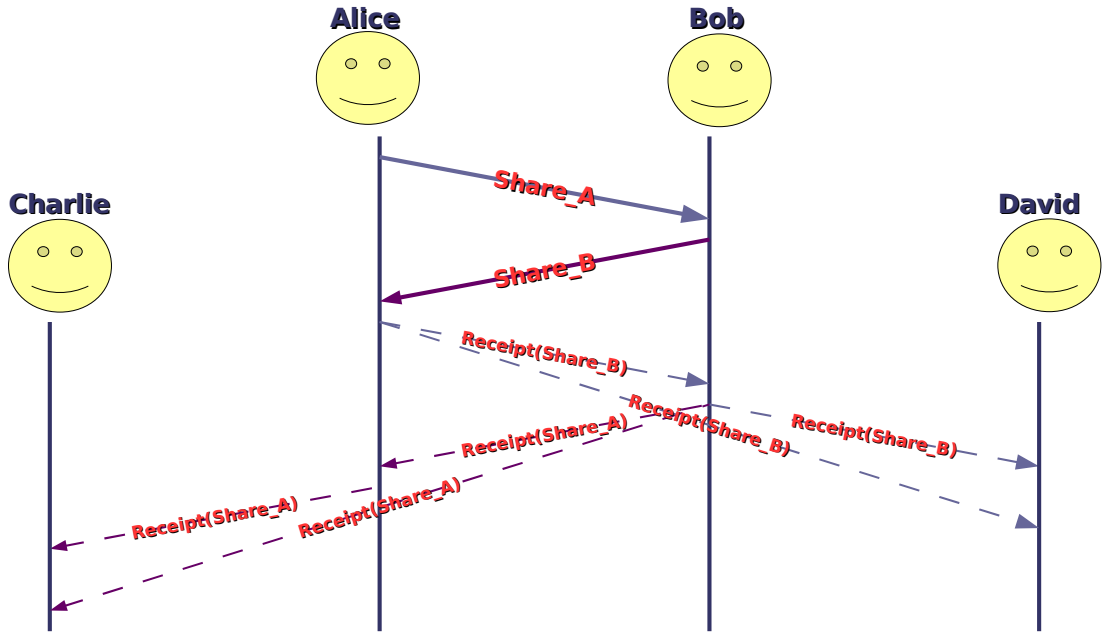

Figure 8: 4-round handshake

When sending a receipt a server makes a commitment to store the corresponding share. But after the first server ('Alice') sends a receipt, the second server ('Bob') could prove vicious and refuse to send a receipt as well. This implies that 'Bob' will probably drop the share and this could lead to unrecoverable documents if this happens more often. One solution to this problem is to let the servers keep the traded shares for a while and so to ensure that this share does not disappear when 'Bob' proves untrustworthy and refuses to keep the share. This will lead to an increased overhead by a factor of two but it will lead to a greatly increased robustness as well. To make sure that 'Bob' does not drop shares all the time 'Alice' should broadcast a complaint to the servnet and hope that the reputation system (→ Sections 3.6 and 3.7) handles it correctly.

3.5 Receipts

As said before receipts are used to check if a trade of shares has been successful. They are not considered as a proof for a completed transaction because a receipt only proves that three out of four steps were successful. But receipts are meant to be broadcasted along with a complaint to fortify that. With this receipt the reputation system can check if a share should be valid and can compute the gravity of this misbehavior, since the necessary information is stored within a receipt. A receipt could look as follows:

```
'I am'       : Alice
'I traded to': Bob
'I gave away': H(PK_[Share_A]), share_num_[Share_A], expiration_date_[Share_A],
               size_[Share_A]
'I received' : H(PK_[Share_B]), share_num_[Share_B], expiration_date_[Share_B],
               size_[Share_B]
'Timestamp'  : timestamp_[GMT]
```

In this case `H(PK_[Share_A])` means the index of the document to which this share ($Share_{Alice}$) belongs to. It is represented through the hash-value of the public key of the corresponding document (PK_{doc}). The value `share_num_[Share_A]` describes which share of the document is addressed. With `expiration_date_[Share_A]` the reputation system can check if a share still should be valid and along with the value of `size_[Share_A]` it can compute the gravity of this misbehavior.

3.6 Accountability

Accountability is important but also very hard to achieve in a system which is committed to anonymity. That this system can survive it is indispensable that malicious servers can be extinguished, otherwise documents can get lost for ever. Accountability in the Free Haven Service will be mainly regulated through the **Buddy System**. Buddies describe a pair of shares which 'look' and 'care' for each other in the servnet. A share 'checks' periodically if its buddy is still alive and reports anomalies to the system. Each share keeps information about its buddy and notifies it when it is moved to another server. This is realized through the receipts (→ Section 3.5) which both are sent to the server which holds the buddy and to the trading servers. That way four receipt are sent during an successful trade, two to the trading servers and two to each buddy-share (→ Figure 8). Thus buddies keep track of each other and assure that a document and also the service can 'survive'. In the data segment the buddy is defined under `<buddynum>`:

```
<share>
...
<sharenum>1</sharenum>
<buddynum>0</buddynum>
...
```

As mentioned above when a share moves it notifies its buddy with a receipt. Due to high latency which depends on the communications channel (→ Section 4) it could happen that in the meantime the buddy has been traded. If this happens the receipt should be forwarded to the new address(→ Figure 9). This new address can be retrieved from the receipt which was sent after the buddy has been traded. In the example which was already shown in section 3.5 the forwarding address is saved in the line `'I traded to': Bob`.

If now a buddy cannot be contacted a possibility could be that it does not exist anymore and a new buddy could be spawned. But this possibility is left out by the Free Haven Project, since it is to be feared that this would lead to an exponential buddy explosion. The buddy could be temporarily offline or the latency could be simply too large.

3.7 Reputation

The reputation system consists mainly of a database which is used to create accountability. In the previous sections it was shown that servers have to be rated for their behavior.

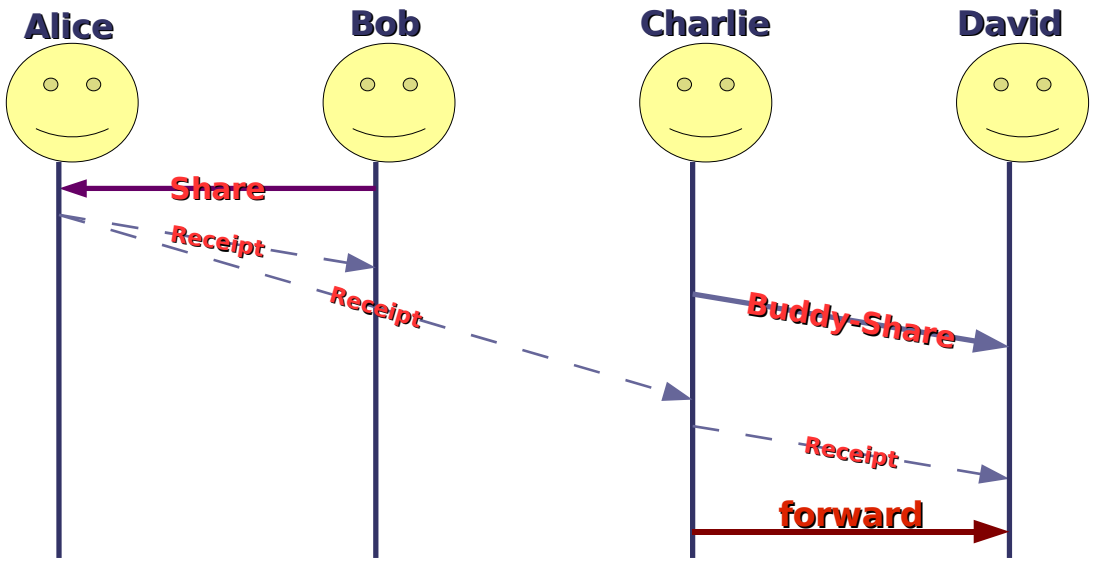

Figure 9: Forward receipt when buddy-share has moved

The reputation system now saves three values for each server, reputation, credibility and the confidence rating. Each server should keep track of these values for each server it knows. Reputation represents in this context the belief that a server will obey the protocol, credibility means that a server's utterances are valuable and the confidence rating expresses the 'stiffness' of these two values.

Servers should broadcast referrals for these values every time:

- a trade was completed, ...
- a buddy gets lost, or ...
- when these values change substantially.

But the reputation system is one of the main reasons why the service has never been realized. Because the solution which was found to this problem is vulnerable to many attacks as well.

3.8 Introducers

Introducers are needed by the service so that new servers can join the network and inactive ones can be removed. If a new server wants to join the service it contacts one of these servers and this one 'introduces' the new server by broadcasting initial referrals. This referrals do not say anything about the credibility of this server, they only indicate that this new server wants to do some trades to increase his reputation. At the beginning when the new server has no reputation it might happen that no other server wants to trade. Then it could be necessary to do one-way trades and offer storage space to the network. To minimize misusage a server can only become an introducer if it enjoys a good reputation. Thus a vicious server which wants to become an introducer first has to act correctly for a long time.

4 Communication

As shown in section 3 the publication system requires the following operations from the communications channel:

- There has to be the possibility to send anonymously messages to a node (e.g. when publishing documents), ...
- send anonymous broadcasts (e.g. when querying for documents or broadcasting reputation values), ...
- name pseudonymously a node within a network for bidirectional communication (e.g. retrieval of a document), ...
- and it should be possible to add nodes to and remove nodes from the communications channel without impacting functionality of the system (e.g. let servers join and leave).

But the communications channel should provide the following criteria as well:

- It should provide timely message transmission, ...
- as well as these messages should be transmitted timely to provide delivery robustness of messages, ...
- when nodes leave the network this should not lead to the loss of the anonymous communication (routing robustness), ...
- it should be resistant to attacks, ...
- and to maintain efficiency, security and reliability the channel has to use a decentralized structure.

Even though these requirements are more than six years old there does not exist a communications channel which provides all of these premises. The problem is to provide anonymous and timely message delivery at the same time. Anonymous communication today is implemented as remailers [4], [6]. These provide good anonymity in the right configuration, but with every level of anonymity gained the latency rises as well. Today remailers delay the delivery of messages to avoid traffic analysis, but this means increased latency as well. That latency can be hours, some sources say it can be even days [1]. So this service is inefficient when it comes to retrieving documents and this is one of the big problems. Since there is no alternative to this without impacting anonymity the service was intended to use a mix between **Cypherpunk** [14], [7], [1] and **Mixmaster** [14], [8], [2] remailers. The mix should be used because the **Mixmaster Remailer** does not support reply blocks which are used for bidirectional communication, but it holds less flaws than the **Cypherpunk Remailer** and so is more resistant to attacks. With the **Mixminion Project** [12], [3] a new remailer was introduced which uses **TCP** [11], [5] and should be more suitable for the service.

5 Conclusion

The first thing that should be mentioned is that the Free Haven Project has never been realized. Even in their paper [13] they mention that the design is unsuitable for wide deployment at that point. One of the major problems is inefficiency.

> "[...], the efficiency and perceived benefit of the system is more important to an end user than its anonymity properties. [13]"

This implies that an inefficient service will lead to few users which will directly impact the anonymity. An anonymous system can only survive if it has a certain amount of users. Thus before implementing the service the efficiency has to be enhanced first.

A low-latency pseudonymous channel is one of the main points why the service is inefficient. When using one of the current remailers the latency is too high for the possibility of widespread acceptance. For an analysis of the provided anonymity and the possible attacks on the system please read the project's paper [13] or the master thesis [14].

The Free Haven Project could not keep its promise. The single goals were not new at all, but no other project tried to provide them at once. One opportunity could be to join efficient peer-to-peer networks. This way these networks could take over the sharing of popular documents which should be accessible quickly and where strong anonymity is not preferred by the user, and less popular documents which do require strong anonymity could be served by the Free Haven Project.

All-in-all you can say that the Free Haven Project failed. Due to the strong anonymity assumption it is necessary to use a communications channel which has a too high latency to be usable in wide scale. And since peer-to-peer networks are considered evil and only used for sharing illegal content there should not be too much backup from the industry or the government.

The future will show if the project will be ever realized. It could pose a possibility to cut back a few goals in order to get the project running and to add features now and then.

List of Figures

References

[1] http://de.wikipedia.org/wiki/cypherpunk_remailer.

[2] http://de.wikipedia.org/wiki/mixmaster-remailer.

[3] http://de.wikipedia.org/wiki/mixminion.

[4] http://de.wikipedia.org/wiki/remailer.

[5] http://de.wikipedia.org/wiki/transmission_control_protocol.

[6] http://en.wikipedia.org/wiki/anonymous_remailer.

[7] http://en.wikipedia.org/wiki/cypherpunk_anonymous_remailer.

[8] http://en.wikipedia.org/wiki/mixmaster_anonymous_remailer.

[9] http://en.wikipedia.org/wiki/plausible_deniability.

[10] http://en.wikipedia.org/wiki/secret_sharing.

[11] http://en.wikipedia.org/wiki/transmission_control_protocol.

[12] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[13] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[14] Roger R. Dingledine. The free haven project: Design and deployment of an anonymous secure data haven. Master's thesis, Massachuetts Institute of Technology, June 2000.

[15] M. Freedman. Design and analysis of an anonymous communication channel for the free haven project, May 2000.

[16] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.