Distributed Mail

A distributed asynchronous message service, is that possible?

Analytical Network Project
Master education in System and Network Engineering
University of Amsterdam, the Netherlands

ing. J.H. Bouwsma
jaap@os3.nl

ing. R.M. Visser
fear@os3.nl

29th June 2004

Abstract

This document describes the possibilities for creating a distributed asynchronous message service. It presents the requirements for an entirely new mail infrastructure on a peer-to-peer based environment. Information about essential topics like privacy, confidentiality and security regarding distributing messages is discussed. Also a review on already existing implementations and research projects for distributed peer-to-peer mail is included.

Contents

1 Introduction

The current Internet has almost reached the limit of its possibilities. Although it is widely used nowadays, the basic design of the infrastructure is already 30 years old. The current email infrastructure dates back 30 years as well and with all the spam these days, the email servers have a hard time dealing with all the traffic.

During the last few years there has been done a lot of research on distributed networks. PlanetLab[1] and OceanStore[2] are good examples of a new generation of networks. These networks are created to provide a strong and robust platform to host all kinds of services. Email could be one of those services and this document provides an analysis of a new kind of asynchronous message system.

The next chapter will present the main research questions about the possibilities of running an asynchronous messaging service in a distributed environment. The main research question will be followed by chapter 3, which starts with the requirements for a new distributed messaging environment. Chapter 4 gives an impression about the current mail infrastructure. In chapter 5 the problems in this current infrastructure are given and chapter 6 gives a more detailed look for an entirely new distributed mail environment. After that chapter, an impression will be given about existing peer-to-peer networks, which already have the ability for distributed mail. The disadvantages of a distributed mail environment are given in chapter 8. This report will end with a conclusion concerning our research objective from chapter 2.

2 Research Objective

The main research question in this report is: "*Is it possible to design a distributed mail-service, which could run in a distributed environment like peer-to-peer?*"

To provide an answer on this main research objective, this document also covers an answer for the following sub-questions:

- What are the requirements for a new messaging infrastructure?
- What are the problems and shortcomings of the current email infrastructure?
- What problems will arise when distributing mail messages in a distributed platform?
- How is privacy and security assured in a distributed environment?
- Are there already existing distributed mail implementations?

3 Requirements for a new mail infrastructure

This section will give an impression about some major requirements for an entirely new asynchronous messaging infrastructure.

[1]`http://www.planet-lab.org`
[2]`http://oceanstore.cs.berkeley.edu/`

3.1 Confidentiality

Messages should be kept confidential. Only the receiver of a message must be able to view the contents of a message. The rest of the world shouldn't be able to read the message, especially if it is sent over an insecure medium. This requires secure transmission and secure storage with strong encryption.

3.2 Authentication & Authorization

A new infrastructure should use authentication to prevent unauthorized mail transports. When a user has some messages queued to sent, he/she needs to give manual permission for the messages to be sent. This system should avoid worms or viruses from hijacking a mail client so it could send lots of messages to random people.

To avoid spam messages, authorization should be needed before a message can be sent to the recipient. This would force spammers to first send loads of messages to ask permission to send emails, which would probably be rejected by nearly everybody. To maintain a user-friendly system it might not be necessary to ask permission for every message which needs to be sent. It should be enough to give your friends, family and/or colleagues permission only once, and they could use this permission to send messages for a longer period until you would withdraw their permission.

3.3 Non-repudiation

If you received a message, you should be able to verify that the sender of the message is guaranteed to be the real sender. One should not be able to sent messages as if they came from someone else (i.e. the principle which also is used for sending spam). This requirement could probably be used in combination with the authentication mentioned above. For each message a person would sent, he/she could give permission for transmitting the message by digitally signing it. This prevents abuse of the user's mail transmission agent and the digital signature guarantees the authenticity of the sender.

3.4 Stability

Besides a secure environment where users can be sure that their data is save and where everybody can be sure nobody else can view their messages, a stable environment is also needed. A secure message system is nothing without a stable environment where users can depend on the availability of the message they send and receive. Therefore, the new infrastructure should provide a stable environment where users aren't dependant on a single point of failure.

In order to keep a stable environment without single points of failure the standards used to provide the stable environment should be enforced. Everything that doesn't work according to the standards should be dropped the very instant it is detected.

4 Current infrastructure

Before describing the shortcomings of the current infrastructure in chapter 5, this chapter will give a global view of it.

The current email-infrastructure can be divided into two parts, namely the processes that exist on the part of the client and the processes that are done server side. First of all a message is composed with a program called a *mail user agent* (MUA). Messages can be composed with the MUA which also takes care of the required message format and the creation of the appropriate headers according to RFC 2822[3]. The MUA will give the message to a *mail transport agent* (MTA) which takes care of the delivery of the mail. The Simple Mail Transport Protocol is commonly used for delivering email. Finally the *Mail Delivery Agent* (MDA) accepts the mail from the MTA and performs the delivery into the mailbox. In figure 1 a graphical view on these principles is included.

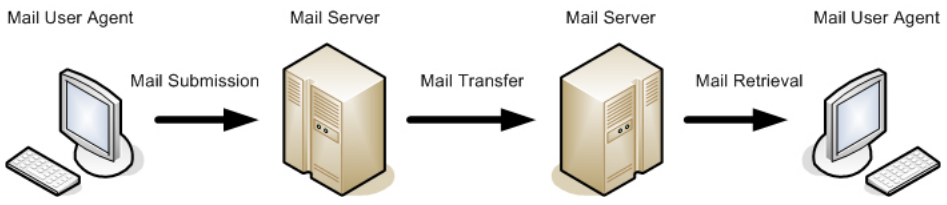

Figure 1: Email protocols in the Internet

4.1 Mail Transmission

The decision on which server the mail-message needs to be delivered is made according to the DNS MX records[4]. A domain might have multiple MX records with different priorities. If the mail server with the highest-priority (i.e. the lower the number, the higher the priority) is not available, the next given mail server according to its priority will be used. This way it is possible to have some sort of redundant mail-delivery if one of the mail servers is unavailable. When multiple MX records have the same priority, one of the server could be chosen completely random, which might be used as a form of load balancing.

The transport and routing of messages between systems is provided by the SMTP protocol [2]. There are many SMTP implementations, including commercial and free/open source ones. Well known open source implementations are Sendmail and Postfix.

4.2 Mail Delivery

The Mail Delivery Agent delivers the messages into the message store. The mail delivery agent might also perform spam filtering and/or virus checking. For the delivery of messages in the message store, it has knowledge about several mail formats like mbox and maildir (see also sections 4.4).

[3]Unfortunately not every MUA uses the strict specifications nowadays [1]. This actually is a bad habit. To assure the compatibility between multiple systems it is important to follow the specifications according to the current RFCs.

[4]MX stands for Mail eXchange

4.3 Mail Retrieval

The retrieval of a mail-message from a remote mailbox is done with a *Mail Retrieval Agent* (MRA). The MRA often works with the *Internet Message Access Protocol* (IMAP) or with the *Post Office Protocol* (POP) and is often included in a MUA.

POP [3] is intended to permit a workstation to dynamically access a maildrop on a server in a useful fashion. Usually, this means that the POP protocol is used to allow a workstation to retrieve mail that the server is holding for it. POP is not intended to provide extensive manipulation operations of mail on the server; normally, mail is downloaded and then deleted. Although it is possible to keep the mail on the server, it is not the main reason for which it is intended.

A more advanced delivery protocol is IMAP [4]. IMAP allows a client to access and manipulate electronic mail messages on a server. It permits manipulation of remote message folders, called "mailboxes", in a way that is functionally equivalent to local mailboxes. Compared to POP, email is stored on a central server which can be accessed by any workstation. In contrast to POP it does not copy email to any local workstation, since a user might have different workstations.

4.4 Mail storage

To provide a new way of distributed mail storage we are first going to have a closer look on the current ways of mail storage. Two popular ways to store mail on UNIX-based mail servers are "*mbox*" and "*maildir*". In this section we will examine both of them.

mbox Mbox stores all e-mail messages concatenated in a single file per folder. A special marker is used to detect where one message ends and the next message begins. With mbox it is not possible to have more than one process accessing the mailbox for writing concurrently, which makes is necessary to use a locking mechanism.

maildir Maildir stores each message in a separated file with a unique name. This makes it only necessary to use locks on a message to prevent two processes from writing over the same message.

5 Shortcomings of the current infrastructure

A new mail system should solve the problems with the current email system. The current problems we identified are:

- Confidentiality
- Authentication, authorization & non-repudiation
- Stability
- Attachments and multiple recipients

5.1 Confidentiality

When the current email infrastructure was designed, the Internet was just a small network between a couple of universities and the term “hacking” was unknown. Current emails aren’t encrypted when they are transmitted and can be read by everyone that can receive or forward the message.

There is however a wide range of encryption technologies available, but these are not enforced by standards. Most popular mail-clients don’t provide these encryption possibilities by default either. Although there are already plug-ins available, which provide for example PGP-possibilities, the user is not enforced to use these. Another fact is that normal users aren’t aware of the fact that there is no confidentiality of their private messages.

5.2 Authentication, authorization & non-repudiation

In the current email infrastructure there is no form of authentication or authorization. It is possible to send messages to other people as if it is send by someone else. There is absolutely nothing done to verify if the sending person really is who he/she pretends to be. Unlike POP or IMAP, which are used to read received messages with username and password authentication, SMTP has no form of authentication. Unless it is an open relay, the only limitation with sending of messages via that mail server is filtering on IP addresses. These IP addresses are usually reserved to just the IP range of the ISP. When a password is needed to sent messages, it gets harder to abuse compromised mail clients.

The current infrastructure also lacks authorization to what messages are sent via a specific account. Every message sent is accepted automatically by the mail servers. When a client is hijacked, the hijacker is free to abuse the client in any way he wants. With authorization the client owner would see all the messages that the hijacker wants to send and could stop all of them, stopping the spread of the virus or worm.

The address from which the message is sent can be faked rather easy and nothing can verify the sender. Non-repudiation isn’t possible with the current version.

5.3 Stability

The central mail server where all the mail-messages are stored could be a single point of failure. When the mail server is down, it is not possible to check for new mail-messages. Also when the mail server gets damaged (for whatever reason) the mail could be lost. Of course it would be possible to have a periodic backup, but this would never be completely up-to-date to even restore the most recent received messages. When email is stored on a P2P network all mail is spread out on the network and there is no single point of failure. With mail stored on multiple nodes the chance of mail being on offline nodes is greatly reduced. A DDoS attack to make a single mailbox unavailable will also fail as multiple nodes need to be targeted at the same time. When information is stored in a completely anonymous way, it would be virtually impossible to know where information is stored and which nodes to target for an attack[5].

[5]This principle is used in the Freenetproject, which is mentioned in section 7.1

Besides the single point of failure another big problem threatens the stability of the current email infrastructure. The fact that the infrastructure supports the abuse of the mail standards, defined in RFCs, is a big problem for the stability of the system. When a message is sent, it should be rejected by anyone else when it doesn't conform to the standards. The current email infrastructure accepts all kinds of violations of the standard and this threatens the stability.

5.4 Attachments and multiple recipients

The current mail standards were created in a period where mail messages where not very large. Those days it was very common for sending plain text messages with sometimes a relatively small attachment. Nowadays it is not uncommon for receiving mails with several large documents, mp3-files or other even larger files attached. Some people expect the size of attachments will still grow in the future, when people will be for example sending video-messages to each other. Every time an attachment is sent it needs to be stored on the email server of the recipient(s). When a newsletter is sent or when somebody is sending an email to multiple recipients, every recipient is sent an individual email. This takes up a lot of bandwidth and storage space, especially when large attachments are sent. If messages were stored on some kind of P2P network, every message only needs to be stored on a couple of nodes for redundancy. Sending a message on a P2P network should save on bandwidth, as the message is spread to a couple of nodes and every recipient could just be sent a link to the message.

6 A new distributed messaging environment

When designing a new distributed messaging infrastructure we have two opportunities. The first one is to design a new implementation of a completely new distributed messaging-network, which is independent from existing peer-to-peer environments. The other opportunity is to design an implementation which could work on existing peer-to-peer environments. We will first take a look at the possibilities for designing our own infrastructure. After that in section 7 we will take a look at the possibilities of an existing platform.

As we have seen in section 4.3 there are two commonly used protocols for the retrieval of messages, namely POP and IMAP. With IMAP messages can be kept available on the server, so users can always access their mailbox, including previously received and read messages, from any location. In a distributed environment it is not very useful to offer users a distributed mail store for messages that already have been read (i.e. a distributed IMAP service), because it could cause problems regarding the available storage space. In a distributed environment there are no real 'home servers' as with the traditional mail environment, so it could be stated nobody is responsible for the availability of enough disk space for messages. All users share in fact one large 'disk' which is shared over several nodes. If however all users in a distributed messaging environment always want there mail to stay available, they require their messages to stay in the distributed environment 'forever' (or at least for a long time). This functionality is very hard to offer in a distributed environment, since it almost requires an 'infinite' data storage. There already seem to be projects which try to implement a gigantic (almost infinite) data store, but one might wonder why they want to have their complete mail history distributed around the world till infinity.

Designing an entirely new distributed messaging environment is not very easy. Not only we have to take care of the basic requirements as mentioned in section 3, but we also have to take a look at a more technical view of the P2P-network itself. In an entirely new design the following questions need to be answered:

- How does 'node-discovery' work?
- How do nodes communicate?
- Which routing algorithm will be used to retrieve new messages?
- How is storage on nodes managed?
- When will messages from nodes be deleted?
- How is security and privacy assured?
- How is 'garbage collection' managed?

6.1 Reliability

In peer-to-peer environments the individual peers might appear to be 'up' or 'down' independently. In contrast to the traditional client/server-model there are no guarantees made about the availability of a 'server' and nodes might show an unpredictable behavior. To provide a more reliable service for storing information, it is required to distribute the information over several nodes.

Let's assume we have a network with n nodes, where the chance that node n is available is p_{a_n}. The chance of the worst-case scenario where all nodes are down would then be:

$$p_{disaster} = (1 - p_{a_1})(1 - p_{a_2}) \cdots (1 - p_{a_n}) \tag{1}$$

In the formula above we take in account that the availability of nodes (p_{a_n}) could differ a lot. Nodes on a DSL-connection for instance would have a better availability than nodes which still use a dial-up-connection. To give a useful estimation about the reliability of a new network, it might be useful to assume the availability of nodes is divided homogeneous. This would give the following simplified formula:

$$p_{disaster} \approx (1 - p_a)^n \tag{2}$$

We also have to take in account that in a P2P-environment not all nodes know *all* other nodes in the network. Providing a so called *full-mesh* (i.e. where each node knows every other node) could give nodes in really large networks a 'nervous-breakdown' and isn't feasible either. Nodes could join and leave the network in an unpredictable way and it is useless to notify *all* other nodes in the network about these changes in the topology. In chapter 4 of [5] a more detailed approach is given for calculating the data persistence.

Instead of a *full-mesh* every node has some 'seeds' into the network, which in turn know other nodes. This way, nodes can be used as some sort of 'bridge' or 'router' to forward requests and information to other nodes. To provide a reliable connection to the network each node has to know at least a minimum number of recent other nodes, to connect to. This number depends on the availability of the nodes and could be derived with formula (2) above.

When a user wants to check if there are new messages, he/she needs to check multiple nodes. Accessing just one node is not enough, since the specific node does not necessarily contains *all* new messages for that person. It is very likely that new messages for a person are distributed over several nodes and that there is not one node which contains *all* new messages. This fact requires a user to check multiple nodes for new messages. To prevent the behavior of nodes becoming too complex, it is not necessary for nodes to synchronize their messages. When a user collects its messages from several nodes, it can perform a check on duplication on its local workstation.

6.2 Security & privacy

Distributing personal messages to several nodes which you don't know and which you can't control involves some major privacy and security risks. In the traditional mail-infrastructure personal mail messages have been traveling through the Internet in plain text. Everyone in the chain from the sender to the mail server could intercept the messages and read the content of it, just like we are used to with ordinary postal cards. However in this traditional way the mail will be stored on a location which you are aware of and (hopefully) trust. In a distributed environment your mailbox could be virtually anywhere and you might lose control of your messages (i.e. when nodes go down or won't join the network anymore). This way it is very important to assure privacy, security and integrity of messages in a distributed environment.

Encrypting the message-body's with PGP[6] could prevent others from reading the content of the messages. PGP also offers the ability to digitally sign the messages, to ensure the integrity and the sender. The disadvantage of this approach is that it only encrypts the body and not the subject and the other headers of the messages. Since the subject is not encrypted eavesdroppers still have the possibility to get an impression about the content of somebody's mailbox. Also the fact that the headers are in plain-text, gives eavesdroppers/attackers on the nodes information about which persons or organizations are sending messages to you. It is even possible to get an impression about the amount of messages someone is receiving. From this point of view distributed messaging involves several privacy issues.

6.3 Scalability & Stability

A new infrastructure must be scalable enough for providing its services to a growing number of users. When more and more users try to spread their messages, a mechanism is needed to prevent nodes from becoming overloaded. To maintain a stable situation of the network it's best practice would probably be for each node having an equally divided number of '*seed nodes*' into the network which is homogeneously divided around the network topology. Otherwise centralized nodes could experience a high traffic and processing load.

[6]A free open source replacement for PGP (Pretty Good Privacy) is GNU Privacy Guard, which can be found on: `http://www.gnupg.org`

7 Using an existing P2P-platform

For deploying a messaging environment on an existing peer-to-peer platform, it forces us to look at the available operations in such a network. The popularity of peer-to-peer platforms still increases and new and better platforms evolve every day. The popular file sharing portal Zeropaid[7] shows a listing of more than 90 popular File Sharing Tools. Most tools are only intended for sharing files and have no other functionality's. Only a few of them contain additional functions like chatting, but none of them shows a really good implementation for asynchronous communication.

Most tools use the same underlying P2P-technology, which has its own characteristics. At this moment the following popular P2P-applications are available:

- Gnutella - `http://www.gnutella.com/`
- eDonkey - `http://www.edonkey2000.com/`
- Kazaa - `http://www.kazaa.com/`
- Freenet - `http://www.freenetproject.org`
- Waste - `http://waste.sourceforge.net/`
- Bittorrent - `http://www.bittorent.com/`
- PlanetLab - `http://www.planetlab.org`
- Oceanstore - `http://oceanstore.cs.berkeley.edu/`
- Entropy - `http://entropy.stop1984.com/`
- GNUnet - `http://www.ovmj.org/GNUnet/`

Most P2P networks do not really have 'intelligent' nodes but only have an interesting protocol for sharing files. They do not have the ability for intelligent file duplication. In these situations the content on the nodes is maintained by the owner and he or she decides which information is stored on the node. Such networks are not useful with asynchronous messaging. For storing asynchronous messages in a distributed environment the node-owner must not have the ability to control the information which is stored on his node. The nodes itself should decide over which nodes messages will get distributed.

The next sections focus on existing P2P-projects which already have some kind of distributed mail infrastructure.

7.1 The Freenetproject

Freenet is founded by Ian Clarke and aims to be "A Distributed Anonymous Information Storage and Retrieval System"[8] [6]. Freenet offers a way for publishing and obtaining information without fear of censorship. To achieve this

[7]`http://www.zeropaid.com`

[8]The original paper called "Freenet: A Distributed Anonymous Information Storage and Retrieval System" was the most cited paper of the year 2000 according Citeseer (`http://citeseer.ist.psu.edu/`).

goal all communications by Freenet nodes are encrypted and "routed-through" other nodes, to make it extremely difficult to determine who is requesting information and what its content is.

Information in the Freenet-network is stored with several types of keys. The key type is included at the beginning of a key, followed by an at-sign (@) and the real information about the specific key. The key types are:

- CHK - Content Hash Key: Every document has a unique corresponding hash value. By using this hash value as its key, two equal documents always result in the same CHK. All other key types usually redirect to a CHK, so almost all information is behind these keys.

  Example:

  `CHK@wQYjaBkpulyIBq4sbvyDL2NZ7ToPAwI,qW4hMEc6NWsx-T-etpfPAg`

- KSK - Keyword Signed Key: These keys can be seen as 'more readable' pointers to CHKs. KSKs do not give any guarantees about security and can easily be 'hijacked', so in fact these keys are deprecated for most purposes.

  Example:

  `KSK@plays/Shakespeare/Coriolanus`

- SSK - Signed Subspace Key: SSKs consist of two parts: one for inserting information (i.e. the private key) and one for retrieving information (i.e. the public key). As the name of the key already says, it provides some sort of *subspace*, in which information can be stored. The subspace could be seen as some sort of 'domain', which is often used as a so called '*freesite*'. Within a freesite it is possible to publish a website, just like we are used to with the current Internet. The biggest advantage of SSKs is all sites in one subspace are guaranteed to be of one author, since only one person owns the corresponding private key.

  Example:

  `SSK@rBjVda8pC-Kq04jUurIAb8IzAGcPAgM/TFE//`

FreeMail FreeMail [7] is an implementation of a peer-to-peer mail server developed by David McNab. FreeMail works on the privacy protecting network Freenet[9] as well as Entropy[10]. These two networks provide the ability for sending and receiving messages with the emphasis on privacy and anonymity possibilities. Entropy stands for *Emerging Network To Reduce Orwellian Potency Yield*, it supports the *Freenet client Protocol*[11] (FCP) to offer full compliance with the Freenetproject.

Each user gets a hash which can be compared to a domain on the Internet. With this hash the user can add names to it to get different email accounts.

Example:

[9] http://www.freenetproject.org
[10] http://entropy.stop1984.com/
[11] http://www.freenetproject.org/index.php?page=fcp

```
Robert@rBjVda8pC-Kq04jUurIAb8IzAGcPAgM.freemail
```

Unfortunately these addresses are hard to remember and there is no service to "bind" a name to the key as these services violate the general principle of Freenet.

Users create their own 4096-bit RSA key pair, which is used to encrypt and decrypt Request to Send (RTS) and Clear to Send (CTS) messages. Whenever someone wants to send a message they need to have permission to do so. A RTS message is sent to ask for permission to send a message and CTS is a reply to the RTS message to grant permission. If two people want to send mail to each other, both need to send a RTS message and a CTS message, so both are allowed to send mail to the other person.

RTS/CTS Protocol A RTS message consists of:

- RTS recipient FreeMail address
- RTS sender FreeMail address
- the URI of an SSK@ message queue, which the RTS recipient asks the RTS sender to poll for subsequent actual mail messages

A CTS message consists of:

- RTS sender FreeMail address
- RTS recipient FreeMail address
- the URI of the SSK@ message queue, which the RTS recipient originally proposed a public encryption key, which the RTS recipient should use for encrypting messages to the RTS sender

Normal message transfer Each sender first needs to have received a CTS message to be allowed sending a message to the target mail recipient. When a CTS message is received and the sender is allowed to send messages he'll need two things:

- An SSK@private key URI prefix to send the messages to.
- A public asymmetrical encryption key, which should be used for encrypting messages to the recipient.

The mail recipient needs two things as well:

- An SSK@ public key uri prefix, for polling for messages from a sender.
- A private asymmetrical decryption key, which she uses for decrypting messages inserted by a sender.

7.2 JXTA P2P-email

JXTA P2P-email[12] is a project from JXTA^{TM} connected technology started at Sun Microsystems. JXTA^{TM} technology offers a set of open protocols to allow any connected device (ranging from wireless PDAs and cell phones to PCs and servers) on a network to communicate in a P2P manner.

JXTA P2P-email is built on the Apache James mail server. Although JXTA P2P-email tries to create a peer-to-peer based mail system, it differs from other projects in the way it does not replicate the user's inbox to a multitude of trusted peers (i.e. there is no distributed inbox). This actually is an interesting different approach, since most other projects try to distribute the users inbox. Unfortunately since 2002 this project does not seem to show much activity anymore, and it also lacks good documentation and specific design notes. The latest (and only) project release we found was the Alpha Release 0.1, which was posted on the 19th of March 2002.

7.3 The Ninja project

The Ninja project[13] is working on a cluster based platform for highly scalable and fault tolerant Internet services at UC Berkeley. The persistent storage is provided by Distributed Data Structures (DDSs), which transparently strips and replicates the data over subsets of cluster nodes. The DDSs take care of the data replication and synchronization, both for the metadata of the Ninja infrastructure and all services build on top of Ninja.

Ninja is implemented in Java and each node runs a single Java Virtual Machine (JVM) that houses the infrastructure unit (a vSpace) and all of the applications on that node. Worker objects are used to house distributed applications. An asynchronous task/completion is implemented to allow communication between workers and between workers and the distributed data structures. Workers are finite state machines that respond to messages passed to them by the infrastructure. This structure has shown to be very effective at improving throughput in systems such as web servers.

7.3.1 NinjaMail

Ninja mail is built upon the Ninja architecture [8] leaving all the scalability, availability and cluster maintenance with the Ninja project, which in turn delegates much of the distribution and data replication to OceanStore (see section 7.3.2).

NinjaMail was developed in two stages. The first stage was a single Ninja cluster architecture to evaluate workloads and consistency of emails. The second stage was a wide area email service based on components of both Ninja and OceanStore. The core of NinjaMail is formed by a MailStore module, which handles all the storage operations, such as saving and retrieving messages and updating metadata. Access modules support specific communication methods between users and NinjaMail, including POP, IMAP, SMTP and HTML modules for user message access.

The MailStore has the following operations:

[12] `http://P2P-email.jxta.org/`

[13] `http://ninja.cs.berkeley.edu`

- addMessage() - adds a message and its metadata to the Ninja DDS.
- getMessage() - retrieves a message from the DDS
- getMessage() - retrieves a list of IDs for messages that match a search specification.
- updateMessage() - updates a message's metadata
- deleteMessage() - removes a message and its associated metadata from the DDS.

Each user has a single "master node" that is allowed to update the mailbox. This simplifies the design, by eliminating race conditions on message index updates. Messages are stored in a distributed hash table and are retrieved by IDs that are generated when a message is added to the MailStore for the first time. Metadata operations use an in-memory database. When a user interacts with the NinjaMail cluster for the first time, all metadata is loaded into the current master node. In the future performance gains can be reached with just loading the inbox in memory. A periodic checkpoint flushes the entire database to the DDS, reducing the number of updates.

The current status of the NinjaMail project is unknown. The last official documentation encountered was from 2000. Some reports were seen from 2002 but after that everything went quiet. Around 2000 a start was made on stage two of NinjaMail and the test cluster of stage 1 was a cluster of 64 Linux servers (dual Pentium III 500 MHz, with 512MB RAM and fast SCSI disks) interconnected with a 100Mbit network. A single cluster with 64 nodes was estimated to service around 100 million users, which shows that the design scales very well [8].

7.3.2 OceanStore

The OceanStore project[14] is developing a global scale information storage and retrieval utility. All data place in OceanStore is replicated across multiple servers. This replication provides persistence when individual nodes fail. The data is protected with encryption, such that only the owner can decrypt the data. OceanStore provides standard APIs to access data. These APIs allow typical read/write actions as well as complex actions as atomic append operations. OceanStore can also migrate data to nodes closer to the clients accessing it, to allow for better latencies.

8 Disadvantages of P2P email

Besides advantages like redundancy and scalability there are a number of disadvantages to P2P email. Server side services like virus scanning and spam filtering are hard to do and in some occasions not possible at all. Email addresses form another problem as there are no such things as domain names. Deletion of messages or files in general is a problem for P2P networks as well.

[14]`http://oceanstore.cs.berkely.edu`

8.1 Virus scanning and spam filtering

It is very easy to introduce and spread large amounts of infected files into popular P2P networks like Kazaa and the eDonkey 2000 network. Because these networks only provide connection and indexing services to other clients, scanning for viruses is impossible in these kinds of networks. When everything is anonymous and encrypted like Freenet, scanning for viruses is completely impossible. The same is true for spam filtering or any other server side service that needs to inspect the data that is stored with encryption.

The only way to scan for viruses and filtering for spam can be done by the recipient. Some kind of mechanism is needed to delete all spam and emails containing viruses from the inbox if the inbox is always saved on the P2P network like NinjaMail or FreeMail. A mechanism like RTS/CTS, which is used by FreeMail, could prevent spam and viruses. A requirement for a successful RTS/CTS implementation requires a stable MUA, which is only allowed to give manual permission before sending out a CTS (i.e. when a MUA is hijacked by viruses or worms they could automatically sent CTS's which still results in unsolicited mail).

8.2 Email addresses in a P2P network environment

A standard email address has a username@domainname format, which isn't suited for P2P email. In a P2P network there are no domains and sometimes no unique usernames either. What a P2P network does have is files and fileID's. These fileID's are usually hashes of the file in question. A P2P email address could just be a fileID.

A possible way to combine a P2P network email system with the current email system and keep the current format is to take the fileID of the email box and combine it with the domain of the P2P network. This way you can make easy translations from the current email address to the P2P email address.

Freenet uses a *username@userhash.freenet* email address format where the userhash acts as a freesite. These addresses however are not compatible with the current email system and you need to be connected to Freenet with your keys to access your mailbox. The biggest disadvantage of a hash as "domain name" is that users can't remember their address, which makes wide acceptation of this kind of email system a lot harder to accomplish.

8.3 Deleting messages

When messages are stored on multiple nodes it gets harder to delete these messages. If a delete order is given every node that has the message stored needs to delete it to completely erase it from the P2P network, but sometimes nodes are down when a delete order is given. This generates all kinds of problems, which results in a different solution to the problem as is seen with Freenet and OceanStore.

Freenet has a simple solution to the deletion problem, you just can't delete something. Deletion is some kind of censorship, which goes against the free nature of Freenet. When a node has reached its maximum storage capacity, an algorithm is started that decides what information is deleted. This decision is based on how popular the data is and how long it's already stored on the node, to avoid new items to be deleted instantaneously as they have never been accessed when they are just put on the node.

NinjaMail has a function called "*deleteMessage()*" to delete received messages. How this function is implemented is unknown. NinjaMail claims to be scalable for billions of users and any method for deletion of items, when nodes reach their storage limit, is unknown.

8.4 Reliability

As seen in section 6.1 a user is not guaranteed to be able to retrieve mail messages which are sent to him/her. Although FreeMail has a mechanism for delivery confirmation, it does not guarantee the delivery of messages. Even when messages are delivered successfully, nodes which store the message can go down and may not come up after a few days. Storing messages on multiple nodes does dramatically increase the reliability of the system.

9 Conclusion

This paper described an answer for the main research question: "*Is it possible to design a distributed mail-service, which could run in a distributed environment like peer-to-peer?*". Using a distributed network like peer-to-peer for providing an asynchronous messaging service is quite an interesting approach. The design of an entirely new mail infrastructure should solve some major lacks in the current email-infrastructure.

The problems are:

- Lack of authentication, authorization and non-repudiation
- Stability issues, because of single point of failure and lack of enforcement of the mail standards
- Lack of confidentiality

Both FreeMail and NinjaMail show a lot of promise when it comes to a solution for the current problems with email. Freenet has extensive features to enforce safe and anonymous mail transfer and a stable environment. NinjaMail provides a high level of security and confidentiality to its users, however NinjaMail lacks the RTS/CTS feature of FreeMail, which could eliminate spam and other unwanted messages.

Distributed email could improve a lot of the security problems that are seen in the current infrastructure. With authentication and authorization implemented, it should reduce the spread of viruses via mail clients, if every message needs to be authorized manually before it's sent. When every message is encrypted and has been signed by the author, messages are confidential and non-repudiation is guaranteed. Because messages are stored on multiple nodes, there is no single point of failure and this could increase the stability of the infrastructure. Stability

Distributed email does however has some weak points (as seen in chapter 8) that will make it hard for the general public to accept it as a new and standard way of communicating in an asynchronous way. Freenet shows the biggest problem, which is the email address. A hash can't be remembered or put on business cards, which makes the exchange of email addresses only feasible through the Internet where the address can be copied and pasted. Reliability is another issue that can play a big part in the wide acceptation of distributed email, however the bigger number of nodes used to spread the same message the more reliable the system becomes. While NinjaMail and FreeMail look very promising, there is still allot of work to be done to get a message system that can replace the current email infrastructure.

References

[1] P. Resnick. RFC 2822: Internet message format, April 2001. `http://www.ietf.org/rfc/rfc2822.txt`.

[2] J. Klensin. RFC 2821: Simple mail transfer protocol, April 2001. `http://www.ietf.org/rfc/rfc2001.txt`.

[3] M. Rose J. Myers. RFC 1939: Post office protocol - version 3, May 1996. `http://www.ietf.org/rfc/rfc1939.txt`.

[4] M. Crispin. RFC 3501: Internet message access protocol - version 4rev1, March 2003. `http://www.ietf.org/rfc/rfc3501.txt`.

[5] Jussi Kangasharju, Keith W. Ross, and David A. Turner. Secure and resilient peer-to-peer e-mail: Design and implementation, January 2003. `http://cis.poly.edu/~ross/papers/email-p2pconf-final.pdf`.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. 2001. `http://citeseer.ist.psu.edu/clarke00freenet.html`.

[7] David McNab. Freemail: Your private and confidential in-Freenet mailserver. `http://www.freenet.org.nz/freemail/`.

[8] J. Robert von Behren, Steven Czerwinski, Anthony D. Joseph, Eric A. Brewer, and John Kubiatowicz. Ninjamail: the design of a high-performance clustered, distributed e-mail system, August 2000. `http://www.cs.berkeley.edu/~czerwin/publications/ninjamail-workshop.pdf`.