TSLab — IT University
Royal Institute of Technology (KTH), Stockholm

Ant: A Distributed Data Storage And Delivery System Aware of the Underlying Topology

Master Thesis

Raúl Jiménez Contreras
`rauljim@gmail.com`

Ant: A Distributed Data Storage And Delivery System Aware of the Underlying Topology

The last version of this document as well as the sources are available at
http://www.tslab.ssvl.kth.se/raul/ant/doc/

Abstract

The distribution of rich digital contents is called to be the next revolution on the Internet. Since broadband connections are available for low prices, there is a great market to exploit.

Nowadays content distribution is performed by P2P –peer-to-peer– systems whose contents are mostly copyright protected. Nevertheless, P2P systems seem to be the way content providers will be able to distribute rich media files universally at low cost, due to the scalability of P2P that client/server model cannot provide anymore.

On the other hand, current P2P systems increase the costs for ISPs –Internet Service Providers– because they are not aware of the underlying topology and hence consume a large amount of inter-ISP width-band.

The goal of this Master Thesis is to design a distributed data storage and delivery system which faces the former issues. It provides a topologically aware lookup service which priorises local transfers and a cache system which allows clients to transfer data from the local network instead of getting it from the Internet.

A complete system has been designed, however, this thesis focuses on the lookup service and data transfer. An implementation of the lookup service has been carried out. This will permit proving the feasibility of such a system as well as detecting scalability shortcomings.

Contents

1 Introduction **1**

2 Background **4**
2.1 Motivation 4
2.2 Existing approaches 5
2.2.1 Direct Streaming 5
2.2.2 BitTorrent 6
2.2.3 Kontiki 6
2.2.4 KaZaA 6
2.2.5 Direct Connect 7
2.3 Properties of the Proposed System 7

3 Description of the System **8**
3.1 System Overview 8
3.1.1 Semantic layer 8
3.1.2 Policy Layer 9
3.1.3 Data Storage and Delivery Layer 9
3.2 Data Layer 9
3.2.1 Opaque Blocks without Semantic Information 10
3.2.2 Block properties 10
3.2.3 Locality Awareness 10
3.2.4 Functionality 11

4 Analysis **12**
4.1 P2P Applications 12
4.2 Lookup and Data Storage Overlay 12
4.3 Topologically Awareness 13
4.4 Distributed Data Storage 14
4.5 Data Transfer 14
4.6 Cache 15
4.7 Choosing a Lookup Overlay 15

5 System Architecture **17**
5.1 System Architecture Overview 17
5.1.1 Ant-hill Global Lookup Server 17
5.1.2 Ant-hill Local Lookup Server 18
5.1.3 Ant Client 18
5.2 Lookup Service 20

5.3 Storage Service 20
5.4 Data Transfer 20

6 System Implementation: Ant Prototype **25**
6.1 Shared modules 25
6.1.1 blockref 25
6.1.2 localmsg 25
6.1.3 blocktransfer 25
6.1.4 hostref 26
6.2 Ant-hill Global Lookup Server 26
6.2.1 Modified Chimera Node 26
6.2.2 Global Network Gateway 26
6.3 Ant-hill Local Lookup Server 26
6.3.1 Global Lookup Server's Client 26
6.3.2 Block Data Base 27
6.3.3 Local Lookup 27
6.3.4 Cache 27
6.4 Ant Client 27

7 Experiments **28**
7.1 Environment 28
7.1.1 PlanetLab 28
7.1.2 Scripts 28
7.1.3 Logs 28
7.2 Tests 29
7.3 Measurements 29
7.3.1 Lookup System 29
7.3.2 Identifying Bottlenecks 30
7.3.3 Measurements To Counter Bottlenecks 30

8 Related Work **32**
8.1 Definitions 32
8.2 The peer-to-peer Model 32
8.2.1 History of P2P networks 32
8.2.2 BitTorrent 33
8.3 Commercial Distributed Delivery Systems 34
8.3.1 Kontiki Delivery Management System 34
8.4 Structured Lookup Protocols 35
8.4.1 Tier0: Key-based Routing API 36
8.4.2 Tier 1: Lookup Protocols Abstraction 37
8.4.3 Tier 2: Applications 38
8.5 Chord 38
8.6 Chimera: An Implementation of Tapestry 38
8.6.1 Tapestry 39

9 Conclusions & Future Work **41**
9.1 Conclusions . . . 41
9.2 Lessons Learned . . . 41
9.3 Future Work . . . 42
9.3.1 Joins and Leaves . . . 42
9.3.2 Commands Options . . . 42
9.3.3 Virtual Local Lookup Server . . . 42
9.3.4 Semantic Layer . . . 42
9.3.5 Policy Layer . . . 43
9.3.6 Data Layer . . . 43
9.3.7 Pre-fetching Feature . . . 44
9.3.8 Contextual Advertising . . . 44
9.3.9 Open Future . . . 44

A Message format **45**
A.1 Local Messages . . . 45
A.1.1 Header . . . 46
A.1.2 Commands . . . 46
A.1.3 Arguments . . . 47
A.2 Global Messages . . . 47
A.2.1 Header . . . 47
A.2.2 Commands . . . 47
A.2.3 Arguments . . . 48

B Messaging exchange **49**
B.1 Lookup Service . . . 49
B.1.1 Client . . . 49
B.1.2 Local Lookup Server . . . 50
B.1.3 Global Lookup Server . . . 51
B.1.4 Two Examples of Publication . . . 52
B.1.5 Three Examples of Lookups . . . 53
B.2 The storage system . . . 54
B.2.1 Client . . . 54
B.2.2 Local Lookup Server . . . 55
B.2.3 Global Lookup Server . . . 55
B.2.4 Example of Storage Requests . . . 55
B.3 Data Transfer . . . 56
B.3.1 Trace of a data transference . . . 57

C GNU Free Documentation License **58**
1. APPLICABILITY AND DEFINITIONS . . . 58
2. VERBATIM COPYING . . . 59
3. COPYING IN QUANTITY . . . 59
4. MODIFICATIONS . . . 59
5. COMBINING DOCUMENTS . . . 60
6. COLLECTIONS OF DOCUMENTS . . . 60
7. AGGREGATION WITH INDEPENDENT WORKS . . . 60
8. TRANSLATION . . . 60
9. TERMINATION . . . 60
10. FUTURE REVISIONS OF THIS LICENSE . . . 60

ADDENDUM: How to use this License for your documents 61

Acknowledgments

Firstly, I have to give thanks to the Sócrates/Erasmus Program. I think everybody should spend, at least, one year abroad. The experience could not be described with words, you have to fell it yourself.

Of course, my neighbors in Sundbyberg must be in this section. I will ever remember our great dinners. Because of them I rarely went to Kista early in the morning! Thank you all.

I thank José Luis Tallón who gave me some ideas to design the architecture of the system and solved some technical doubts about networking.

Javier Romero stand my speeches about P2P and the Ant system. Moreover, during July when everybody was on vacation but he and me were working on our respective theses. Thank you for your infinite patience.

I must give thanks to my tutor Björn Knutsson who has guided me to find out the way to face each issue of this thesis. I hope he is happy with the result.

No existen palabras suficientes para agradecer la confianza depositada y el apoyo incondicional que mis padres me han ofrecido en todo momento. A ellos les debo todo.[1]

[1]There are not words to give thanks enough to my parents. They have trusted and have supported my projects every time. Everything what I do is because of them.

Chapter 1

Introduction

The goal of this Master Thesis is to design a distributed data storage and delivery system. This system is based on peer-to-peer (P2P) model whose users transmit the data to each other while the publisher keeps the control of the distribution.

The main object of our system is to design a protocol which takes advantage of spatial and temporal locality. In order to achieve the former, connections between users within the same network are priorized and caching is used in the latter. This system tries to reduce costs for ISPs by cutting down traffic between ISPs remarkably. At the same time, users are rewarded with a lower latency and higher download speed.

Content distribution companies have helped authors to spread their works. The control over the distribution is the key of their business. This control is easier when making a copy is expensive, such as a painting, a sculpture or a book. However, we live in the digital age and the cost of making and distributing a copy is very low.

Nowadays, content distributors are worried about the parallel distribution of their contents on the Internet. Millions of people download films and songs for free, without the permission from the owners. However, the judicial fighting against peer-to-peer users seems to be ineffective since these networks grow continuously. Then, how could the illegal file sharing be stopped?

People demand a good delivery service on the Internet, as iTunes' success shows. If content distributors offer a good service, lots of peer-to-peer users will use it instead of his favourite BitTorrent program. It suggests that content providers will keep the control of the distribution as long as they offer the best service; i.e. while users do not have reasons for spending their time on: cracking anti-copy systems, making and distributing copies of copyrighted material.

Competing against free peer-to-peer model is not an easy issue. However, content distributors are able to offer services that free peer-to-peer distribution cannot. The Ant System tries to exploit this kind of features for clients, ISPs and entertainment companies.

This thesis tries to describe a system where anybody is able to publish digital contents without the need of having powerful servers on the publisher's side and even without previous permission from any component of the network. In fact, publishers only have to publish the content on the system and to offer small files, which contain information related to the actual files, –on a virtual shop, as a part of the program interface, etc–. The hard part of the distribution work –i.e. the actual transmission of the data– is already done by their clients. Their hardest work will be to persuade users to download their contents and, that they actually download them from their official site.

This kind of distribution could be compared with satellite TV. Several satellites send waves to the Earth, however, only the client who has the right decoder can watch the shows on his TV. This system gives everyone the power to broadcast thought the biggest and universal satellite –the Internet. Furthermore, it permits the publisher to choose who get access to the published content by using virtual decoders –small files purchased on a virtual shop–. However, the Internet and computers offer more features and they are more flexible than satellites and televisors.

The Ant System introduces the *pre-fetching* feature. It makes possible the distributed storage of fashionable contents –such as TV series or movies– before their premiere.

The content is downloaded by several computers whose users cannot access it. On the premiere's day –when the publisher offers the small file that decipher the content– clients who get it from the virtual shop are able to watch it **instantly**; thousands, even millions of clients at once.

Pre-fetching shows promise as a very profitable feature due to film's box office trends towards being logarithmic within the time, as Figure 1.1 shows. Since marketing is focused on first weeks, it makes the film fashionable and everyone wants to get it as soon as possible. After a while, another film becomes fashionable and the former gets obsolete. *Pre-fetching* exploits the very early age as it is explained during this paper. From the ISPs'

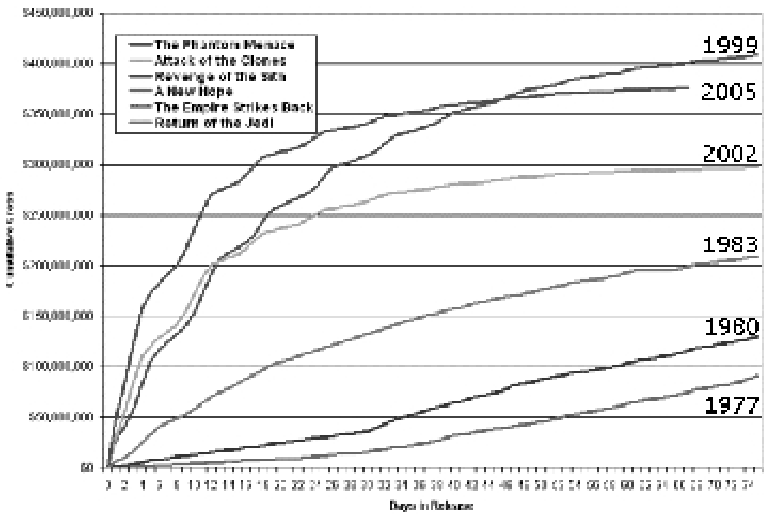

Figure 1.1: *StarWars' box office*. It shows current films' box office are logarithmic functions. (source: http://www.the-numbers.com/features/starwars.php)

point of view, current peer-to-peer systems are causing excessive network traffic. In fact, there are some ISPs that are using several techniques in order to identify and reduce P2P traffic –they are shown in Section 2.2.2. However, none of these techniques are result of the cooperation between the ISP and its users. In fact, there is an interesting debate between developers of BitTorrent clients about the convenience of encrypting connections to avoid traffic shapers.[1]

[1]Bram Cohen, designer of the BitTorrent protocol
http://bramcohen.livejournal.com/29886.html
—
SLYCK - BitTorrent End to End Encryption and Bandwidth Throttling

On the contrary, the Ant protocol offers a symbiotic relationship between them. The system encourages ISPs to put a lookup server and a cache near their Internet gateway in order to route and cache their users' data. This system gets information about network structure and avoid inter-ISP traffic as much as possible.

Since all transmitted data is encrypted users do not need to be worried about their privacy since ISPs are not able to look at the contents their clients manage. Despite that, any ISP can prevent access to a set of contents –e.g. a company could limit the lookups to only a set of contents– as a http filtering proxy does currently.

http://www.slyck.com/news.php?story=1083

Chapter 2

Background

This Chapter provides the motivation of this thesis: which needs it should meet and why current systems are not able to achieve these properties.

2.1 Motivation

The main motivation of building a new system is to satisfie the needs of the actors involved in the rich media distribution on the Internet –content providers, Internet service providers and clients– whose demands were presented in the Introduction. A schematic list of demands is listed below.

- **Content Providers**
  - *Low-cost distribution system* which minimises the need of having powerful servers, a wide connection to the Internet and a huge amount of network traffic.
  - *Keep control over content distribution* in order to limit it and/or for accounting measurement.
  - *Integrity and confidentiality of the content.* Nobody can change a content published and it is not accessible until the publisher wants.
  - *A scalable system* that is able to manage as many clients as possible at once without a worsening of the quality of service.
  - *Open platform* where it is possible to develop specific features. It is also possible to make out-of-box products.
- **Internet Service Providers**
  - *Costs reduction.* An ISP wants to reduce inter-ISP traffic and traffic peaks as much as possible.
  - *Open and cooperative platform* where each ISP has some control over the protocol and can set parameters in order to improve the service and reduce costs.
- **Clients**

- *Easy to use*. The user's program should be simple and hide the P2P layer. It could include a search engine and RSS-like subscription to contents –e.g. a TV serie–.
- *Fast*. Users want to watch the content as soon as it is published. They also do not want to wait several hours –even days– to watch a purchased movie.
- *Cheap*. Hopefully costs reduction for content providers would provide contents at low price. Advertisement addition could generate incoming enough to offer contents for free.
- *Open platform*

  Besides downloading contents, every user is able to publish his o her own content.

2.2 Existing approaches

In this section, the main characteristics of existing distribution systems are shown.

2.2.1 Direct Streaming

It is also known as *client/server* paradigm. Where the publisher uses its own resources –i.e. storage, computer power and network infrastructure–.

FTP servers is an example of these kind of protocols. Mirrors simulates decentralisation around several countries. This schema is widely used to distribute software.[1] Usually universities offer their resources to mirror free software.

Some content/software providers offer their contents through companies like Akamai. Akamai has a huge amount of servers around the world, their clients pay to Akamai according to the amount of traffic generated. This system offers fast downloads, however it is really expensive for publishers.

YouTube, Google Video and other similar services use the client/server paradigm. Nowadays, they have become very successful. It requires a large amount of servers around the world and huge amount of network traffic. On the other hand, servers' owners have the absolute control over the contents and is able to block/discriminate contents or just shut down the service. It is not suitable for DVD-quality nor large video such as films because the costs involved are too high to offer this kind of contents for free.

Neither of these systems are scalable since they cannot manage an unexpected number of clients at once.

Steam

Steam[2] is a client/server distribution system for games. The user installs a client that connects to official servers to download new games as well as updates. There are not technical details of this system, however, its most interesting feature is *preload*. It distributes large files during days before the official release day. At that moment, every gamer can play the game at once. The game provider takes advantage of the gamer's

[1]Debian's mirrors: http://www.debian.org/CD/http-ftp/#stable
It is also distributed through P2P systems: http://www.debian.org/CD/torrent-cd/

[2]http://www.steampowered.com/

wish of being the first one playing the new game. Half-Life 2 was released in this way.[3] http://www.steampowered.com/marketing/hl2_preload/english.html archive.org

2.2.2 BitTorrent

BitTorrent –as the P2P protocols in next subsections– makes clients download contents from each other. That means publisher does not carry every content transfer which, obviously, reduces dramatically distribution costs for publishers. It is scalable as long as the lookup service is.

BitTorrent is widely used for file sharing of copyrighted content without author's permission. However, it is also used to distribute free software –e.g. GNU/Linux distributions–, free music –e.g. music under Creative Commons[4]–, etc.

BitTorrent offers a system where anyone is able to publish any content easily and in a cheap way. Publishers only need to make available a copy of the content, make a small meta-info file (torrent file), register it on a tracker and publish the meta-info on a web site. Users get the torrent file, and download the content with their BitTorrent client.

Integrity is checked by hash functions but content is not encrypted. Contents should be encrypted before when confidentiality is required –or to prevent access–.

A CacheLogic's study [2] claims a third of the total traffic on the Internet is generated by the BitTorrent protocol; ISPs could reduce a huge amount of inter-ISP traffic by improving locality and caching contents. Actually, CacheLogic Ltd. and BitTorrent Inc. have signed an agreement to provide BitTorrent protocol caching feature.[5]

See Section 8.2.2 for further information.

2.2.3 Kontiki

Kontiki is focused on the control of users, publishers and contents. The system relies on secure connections over HTTPS and DRM –Digital Rights Management– to control everything within its Kontiki network.

The distribution is a hybrid between client/server and P2P models. A set of servers is the core of the system, however, users upload pieces of content in order to reduce core's costs.

Since the owner of the core controls everything, it is designed to be a single-company system instead of an open platform where every content provider can publish.

See Section 8.3.1 for further information.

2.2.4 KaZaA

KaZaA uses the FastTrack protocol which is closed, encrypted and undocumented. Some information about the protocol was obtained by reverse-engineering.[6]

In FastTrack protocol there are nodes and supernodes. Nodes that have enough processing power and fast network automatically become supernodes. They index users'

[3]http://www.pcauthority.com.au/news.aspx?CIaNID=16103

[4]http://www.jamendo.com/

[5]CacheLogic and BitTorrent Announce Strategic Partnership to Advance the Adoption of Peer-assisted Content Delivery — 7 August 2006:
http://www.cachelogic.com/home/pages/news/pr070806.php

[6]http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=HEAD&content-type=text/vnd.viewcvs-markup

shares, perform search queries and other undiscovered features such as statistics. This architecture improves scalability, however, it relies on unknown hosts –supernodes are users' computers– reliability.

KaZaA's integrity verification is too weak, this caused a large amount of fakes which frustrate users' experience. The instalation of adware[7] and spyware[8] also bother users who moved to other P2P protocols.

2.2.5 Direct Connect

Direct Connect protocol[9] utilises discovery service instead of lookup service. Each client is connected to a discovery server –called *hub*– which receives requests and broadcasts them. In fact, it works as a physical network hub. Clients which have the file requested reply to the the client. The traffic generated by a request gets greater as the number of clients grows.

2.3 Properties of the Proposed System

The system designed should use clients' storage and network resources as much as possible –i.e. as much as clients allow– to *reduce distribution costs*. At the same time, it should priorize local transfers to *reduce inter-ISP network traffic*; caching will be used to keep a copy in the local network.

Publishers will be able to *keep control over content distribution*. Encryption should assure the *integrity and confidentiality* of the data. It should make it possible to implement accounting and access control systems on the top of Ant.

The system should *permit any configuration on the ISP side*. However, every client will be able to circumvent the policy established by its ISP –it is a requisite to avoid censorship–. Every host will be able to connect to the network and perform any role –lookup server, publisher or consumer–, even several of them at once. These requisites provides an *open platform*.

[7]"**Adware** or advertising-supported software is any software package which automatically plays, displays, or downloads advertising material to a computer after the software is installed on it or while the application is being used." (source: http://en.wikipedia.org/wiki/Adware)

[8]"**Spyware** refers to a broad category of malicious software designed to intercept or take partial control of a computer's operation without the informed consent of that machine's owner or legitimate user." (source: http://en.wikipedia.org/wiki/Spyware)

[9]Protocol documentation: http://sourceforge.net/docman/?group_id=36589

Chapter 3

Description of the System

3.1 System Overview

As said in Section 8.2, peer-to-peer systems are distributed systems whose nodes share their resources and each one employs other's resources to achieve its objectives. In the Ant system, the resources are: storage, network bandwidth and computing power (lookup service).

The aim is to design a stable and scalable system whose users store and download digital contents. Information about users and blocks is managed in a decentralized way to avoid inconsistency when any component leaves or joins the network. In fact, the normal state of the network should be able to bear multiple joins and leaves at the same time. It should manage an unlimited number of users and blocks without a worsening of the quality of service (i.e. it must scale).

The Ant system consists of three independent layers: semantics, policy and data.

Each file is divided in several parts, each part is a block. Each block has a unique ID, which is the result of hashing the length and the content of the encrypted block.

The semantic layer manages information relative to the actual file: name, type of file, length, author, etc. as well as the list of blocks that form it. The policy layer defines the local configuration –caching and storage– and some lookup features such as sort the results list to offer firstly the better host to download from. The data layer treats each block as an opaque box, it is because there is not any kind of semantic information in this layer; its duty is to carry out publication, lookup and transfer functions.

Although this thesis focuses on the data layer, the basic design of the whole system must be made. That is why a brief description of the two higher layers are shown in this section. They might help to understand the whole system.

3.1.1 Semantic layer

Each file has a semantic file where the information about the IDs of the blocks which form the file is stored, their length and information about the file –name, author, creation date, etc.–. This semantics file looks like a .torrent file ; however the information about the tracker is not necessary anymore. This is due to the fact that each client uses the same tracker in all of the lookups –hopefully its ISP's lookup server–.

Moreover, a key is necessary in order to decipher the blocks. This key could be in the semantics file or in a different file.

For example, a user, who wants to transfer private data to another user, will make a semantics file send it solely to the receiver; in this case the key might be inside the semantics file. However, there are other scenarios where the other option could be useful. For example, a recording company which publishes the blocks on the Ant network and the semantics file on its web page. Everyone could download and *share* copies of the blocks, but they will have to wait to the official publication to get the cryptographic key and listen to the music.

Although block encryption is useful to protect users' transfers, it is not a practical solution for entertainment companies which want to lock the contents. In these cases, DRM-based technology[1] should be used upon the semantic layer.

Of course, customers would be happier if companies offer digital contents without locks and for free. These companies would spread their contents faster, take a great piece of the market and make money from advertisements and merchandising.[2]

3.1.2 Policy Layer

The policy layer is a very important part of the system; it will be the target that ISPs must focus on. This is because this layer manages contents storage/caching and that is the key to cut down the inter-ISP traffic. ISPs will cut cost as much as better policy configuration they develop.

This layer is the responsible for caching and the distribution of blocks among its clients; therefore it is able to choose from which source the data will be downloaded depending of technical issues such as latency or speed as well as non-technical such as commercial and peering agreements. However, it could be used for other future features. That is why its design must be open and flexible enough. For example, cache could be configured to accept or reject blocks depending on the source, size, number of requests of that block, etc.

On the other hand, it might be almost transparent for end users. Usually, they will not like to deal with this kind of details they will just want to publish, store, lookup and download contents. However, they could use some optional settings in order to improve their usage experience.

This thesis does not pretend to make a definitive design of this layer. Despite that, the data layer must be designed in order to be independent of the other layers and support a wide range of policy layer's future features.

3.1.3 Data Storage and Delivery Layer

The data layer is at the bottom of the system. It manages everything what is related to block's storage, distribution and data transference. This thesis focuses on this layer and most of the chapters are related to it.

3.2 Data Layer

The main points of the data layer are described in this section.

[1] $http://en.wikipedia.org/wiki/Digital\_rights\_management$

[2] BusinessWeek. 11 April 2006. "Disney's Internet Adventure".
"The media giant will allow users to download popular TV shows from ABC.com. But there are still glitches to work out."

3.2.1 Opaque Blocks without Semantic Information

The Ant system separates data and semantic information clearly.

Each block is unique and independent to the others. They only have the data, no information about where or when they were created or published. It encourages the concept of network neutrality due to every block is managed in the same way. Discrimination becomes hard because there are not parameters to discriminate. It makes it difficult to block contents from a specific source –e.g. a non-democratic government's network against a opponent association or an ISP against an entertainment company that has opposite business interests–.

The publisher can spread its blocks without worrying about how many copies are on the network or where they are. In fact, the more copies on the net, the faster the client's download will be.

3.2.2 Block properties

The system does not enforce any specific block size. Actually, this parameter is chosen by the publisher.

Blocks are named by the combination of size and content hash . This allows any storage node to recompute names and to easily verify the integrity of the data.

Every file is encrypted. In this way, only those who has the key will be able to see the content. Content will be a secret as long as the key is a secret. It is useful when a publisher pre-fetch caches but he does not want anyone watch the content before official publishing date.

It could help to encourage users to give some control to ISPs. Since ISPs do not see what they are sharing unless semantics file and key are avaliable. Of course, ISP can censor a list of blocks but actually it can block any kind of P2P traffic as well. however, since this system reduces ISP costs it is likely it will avoid users to utilize another P2P protocol. Hopefully, ISP will not censor blocks unless it has to do it due to a court order.

On the other hand, users want their downloads to be fast. Because of that they have chosen eMule and BitTorrent instead of Freenet. They usually prefer speed to anonymity. And as it was said before, ISP is able to look at every user's package.

Blocks are encrypted regardless at another encryption systems –any kind of DRM– is used. This one will be a top level protection apart of the system itself.

3.2.3 Locality Awareness

This system should be locality aware. Requested blocks should be download from the network where the downloader is, when available. Cache is utilized to keep a copy on the local network. It must decide whether block is cached according to the caching policy.

If there is not a copy avaliable on the local network, it should find the better block's copy location –in some measurement such as latency or traffic cost–. This is a strong point because ISP can choose the best option every time, it could priorize transfers from/to other ISPs which signed peering agreements[3]. I.e., the better an ISP configures its lookup server the more traffic costs reduction.

[3]"**Peering** is the practice of voluntarily interconnecting distinctly separate data networks on the Internet, for the purposes of exchanging traffic between the customers of the peered networks." (source: Wikipedia)

This is similar to DNS, where users usually connect their ISP's DNS server because of some advantages such as low latency. In this case, their advantage would be to find the closest client where download the data from.

3.2.4 Functionality

This is the API of the user's side:

- PUBLISH any block on the network in order to make it possible that anyone else gets it.
- UNPUBLISH any block which was published by this client when it stops offering it.
- STORE any block on the network, either on a specific host or in a distributed way.
- LOOKUP any block on the network.
- DOWNLOAD any block that was published on the network.

The API is explained in Section B.1.1 as well as in the *ANT Documentation*.

Publication

Every user on the network is able to publish any block –either it is the original or a replica– and from that moment, everyone on the network will be able to download the data of that block from this host.

Storage

Every user on the network is able to offer and demand distributed storage. The capacity of that is managed by the policy layer, the data layer will give it the tools to do that.

Lookup

Every user on the network is able to find where a block's data is when the block was published previously. The system should find the closest set of replicas of the data.

The Ant System will offer a common lookup interface which content providers, ISPs and users can use it as they like. In any way it imitates DNS[4], although the Ant system looks up some host addresses given a blockID while DNS looks up host addresses given a host name.

Data Transfer

Every user on the network is able to download a block's data once it knows which users have a copy of it. The policy layer will manage download requests and ask for other blocks in exchange. It could be performed by a module similar to BitTorrent's negotiation protocol.

[4]Domain Name System

Chapter 4

Analysis

The original idea of this system was to design "distributed data storage and delivery system". However, since users are involved in the system, it should not treated as a conventional storage system where each node is controlled by a responsible part and all together work in order to offer the best service. Since each user can do anything on the network it has to be sure that users are not disturbed. This is a very important point, users really should like this protocol because it is much better than others; otherwise, they will be still using their favourite file sharing protocol.

Therefore, the Ant System has been designed considering users needs. This design tries to make it possible to give each part what they want, because if it achieves that, nobody will need to cheat the system in order to get more from it. The advantages to the users, distributors and ISPs were described in Section 2.1.

4.1 P2P Applications

In the beginning of the reading stage some papers about peer-to-peer file sharing protocols were read. They have some characteristics in common; however, each one was created to cover any particular need. Some of them focus on performance whereas others provide privacy or anonymity by sacrifying efficiency.

Since this project focuses on efficiency and reduction of network traffic; protocols like BitTorrent are very interesting to investigate. However, papers about FreeNet and others focused in other fields gave me some ideas to face this project.

BitTorrent seemed to be a good start point due to its popularity and its principal characteristics. The semantics layer of BitTorrent inspired the semantics layer of this project. However, the separation between semantics and data is stricter in the latter as is shown in Section 8.2.2.

4.2 Lookup and Data Storage Overlay

After taking an overview of more representative protocols in each kind of peer-to-peer file sharing protocols, distributed lookup protocols were studied. The principal drawback of current file sharing protocols is that they need a server in some sense: servers in eMule, trackers in BitTorrent. Although there are protocols which don't need any kind of server

–such as FreeNet– they have serious problems of scalability due to they use flooding techniques to find the data.

There are several groups of developers that are working on distributed system to avoid the need of central servers which perform the searches. Actually, it is possible to utilise a distributed version of the BitTorrent tracker –without a central server running a tracker–. This version uses a Distributed Hash Table (DHT) protocol called Kademlia [9].

Kademlia came very interesting because it seems easy to understand and use. Since it is based on Chord, several papers about it were also read whose main features are discussed in Section 8.5. Although this kind of lookup protocols seemed to be useful for this project, there were some important differences between what Chord does and what this project needs. The principal one is that Chord stores the data in a node on the network when a block is published; it is useful to develop a distributed file system such as [5] because it provides high degree of fail resilience tolerance by replicating data on several nodes.

However, this project needs another kind of lookup system. This is due to these reasons:

- Users can join and leave the network in an unpredictable way. The movement of data could be too intense due to these events.
- Each time a user joins the network it is responsible for some blocks. That means, these blocks should be downloaded and stored on the user's computer. Some users would not be happy waiting some minutes before downloading something they really want to or, even more important, storing unknown data.
- In this scenario is very difficult to configure the Chord network in order to priorise local transfers. It only can choose between a limited number of replicas.

Other DHT protocols were studied, as it is shown in Section 8.4. However, they share the same concepts and, even, the same API [4].

4.3 Topologically Awareness

At this point, researching was focused on how to take advantage of locality in the lookup system. Several papers about topologically aware techniques were read, the most significant papers are briefly described later. Some hierarchical lookup protocols were studied and gave us some ideas about how to use these concepts in this system.

The main characteristic of these protocols is the concept of group. The entire network is divided in groups, each one has one or more *super peers*. As it is defined in [6]: "the super peers are gateways between the groups: they are used for inter-group query propagation". Although this is a good idea, it has several problems:

- Firstly, super peer demands process power and network resources; users may not like to be a super peer and they could try to cheat the system to avoid it.
- Since a super peer is a key piece on the network, it could be a bottleneck –or even it could perform a denial of service attack.
- It is not a good idea to trust in an anonymous node. It looks at your queries and can block some queries or give users wrong information.

Nevertheless, it arose the principal idea of this Thesis. It would be nice if my ISP manages the super node. However, two topics have to be considered:

- Users do not like anybody to control the contents they download. They could reject giving too much control to their ISP. It is the reason why users should have the ability of circumvent it and use another super peer; of course, if they detect their ISP is cheating them –e.g. blocking contents– they will use another protocol. Any way, this system does not give ISPs too much control since they can look at the packages which through their networks. This system makes cheating to be a bad option for ISPs.[1]
- An ISP will not spend money on machines and administrators unless the new system is profitable enough. It should reduce dramatically the amount of traffic between its network and the Internet. And if this is a notable reduction, the ISP will do its best in order to get better this service for users who will be happy to use this protocol instead of another.

4.4 Distributed Data Storage

Distributed Hash Tables are designed to support distributed data storage systems. Actually, several implementations of this kind of systems exist, such as CSF [5] (Chord) and OceanStore [8] (Tapestry).

This feature could be used separately or combined with the whole system. It could be used as a backup system by having several Ant-hill servers in different locations or signing an agreement between companies that want to have copies of their data in different locations; it is possible due to the data is encrypted.

When the publisher's aim is to broadcast the content, this feature could be used in prefetching phase. It would be likely to cache a movie around the world some days before the premiere. Once the premiere's day comes, clients will be happy to download it from a near mirror. At the same time, the publisher will not have to face the whole peak of requests.

4.5 Data Transfer

There are two options to transfer data from a computer to another:

- Make a simple protocol from scratch.
  It will be efficient and offer on-demand features.
- Use a standard protocol.
  It could be less efficient, however, it is more stable and it is easier to develop and maintain.

There are many client-side libraries for several protocols. One of them is *libcurl*[2] which is a multi-protocol client-side URL transfer library used by many network applications. And it could be used in this system due to it is free software[3]. On the server side, a light FTP or HTTP server would manage the transfers.

Despite the fact using a standard protocol is easier to implement, an custom-made protocol is used in data transfers in this system. This is due to the whole protocol is more homogeneous and it was easier to implement.

[1]Note here that censorship is not economical for ISP, however, it could be imposed by judicial and/or political reasons.

[2]Lib curl's home page: http://curl.haxx.se/libcurl/

[3]See libcurl's license at http://curl.haxx.se/docs/copyright.html

4.6 Cache

The concept of *cache* is widely used in computing to reduce the access time and transfer costs. When the cost of storing a copy is less than transferring it again, caching is a good business. In this system the cache is optional, however, an ISP would reduce the amount of inter-ISP traffic by caching some contents.

This system does not enforce any cache's policy, each ISP will be responsible for its own cache's policy. However, the inclusion of a caching management module could be a future task.

4.7 Choosing a Lookup Overlay

Although this section could be seen as a implementation issue, the election of a lookup overlay is a design decision. The whole system depends on lookup overlay's characteristics. This is why this section is here.

Despite Tapestry looks more locally aware, firstly Chord was choosen due to its simplicity. It seemed to be easier to modify Chord than the other because Tapestry implementation seems to be more complex.

As it was said before, DHT was designed for distributed data storage. It stores the information on some node on the network. However, the behaviour that this system needs is only to find where the data is; only who publishes and who downloads a block will store the data. No user will be asked to store anything. Moreover, when a user joins or leaves the network no transfer of data will be needed.

At first, the modification seemed to be easy. The lookup service should store a list of nodes which store the data, even it only has to store the list of super peers. However, Chord –and the others DHT protocols– are designed to manage immutable blocks of data. These systems replicate and cache blocks of data assuming they will not change. This is the behaviour that has to be modified in this project, because our system will store a list of nodes where actually the blocks are stored. And that list is, of course, dynamic.

Since Chord seemed to be the most simple lookup system, it was found out the way to modify it. One of the objectives of Chord is to distribute a block as much as it is solicited. It is possible because of the cache copies and replicas. One problem here is upload the copies of the master list; that is not a dramatic problem because it could be configured in order to get the master list in case that the client is not satisfied with the cached list. Another unlikely behaviour of Chord is that it is not locality aware; nodes which are nearby in the Chord ring could be in different continents.

Tapestry was the other lookup system candidate for this project. Although it is more complex than Chord, it has some likely characteristics. The obvious one is that it is topologically aware; nearby nodes on Tapestry network are close in terms of latency. Although Tapestry is similar to Chord, its behaviour is closer to what this project needs. In a Tapestry's publication the data of the block published is not copied; a reference of publisher node is stored on each node in the path. For example, when a node in Stockholm publishes a block and the root of that block is in USA the path could be: Stockholm, Berlin, Paris, London, New York; each node in the path will store a reference of the node in Stockholm. When a node in Spain lookups that block the path to block's root could be: Madrid, Barcelona, Lyon, Paris, London, New York. As you can see it is not necessary to reach the node in USA, the node in Paris will route the lookup to Stockholm and this node will send the data to the node in Madrid.

The example above shows the behaviour of Tapestry. It is interesting because a publication does not involve data copy and there are references to the node which stores the data. The difference between Tapestry and this project's requirements is:

A Tapestry node does not return a list of nodes which store the block, it route the lookup message to a node which published the block requested. Figures 5.1 to 5.7 show how the publications are done.

This method, known as DOLR (Decentralised Object Location and Routing) does not give the chance of select the best source to download the data. Even worse when a node publishes a corrupted block, the lookup will be routed to the cheater node. In order to avoid this kind of threats, the block is published in different roots but this does not counter totally the threat because the cheater node can also publish on different roots.

Therefore, the modification necessary to adapt Tapestry to this project is based on storing a list of references on each node and returning this list instead of the data in lookups.

There are available two implementations of Tapestry, one in Java and the other in C. Since the implementation in Java seems to be more stable than the other, I read its documentation. It was quite hard to understand because of the multiple options of configuration. The implementation was too large and I had to spend all the time just modifying the code. In spite of the fact, I tried to understand the code and find the way to use it. When I tried to compile the code on my computer, I had several problems with Java's compilator. After trying different complilators, I asked Tapestry's developers why I had these problems. Ben Y. Zhao replied the message, he suggested me to use the C implementation of Tapestry (Chimera) although it was under development at that moment (version 0.8) and some features had not been implemented.

Chapter 5

System Architecture

The design of the system architecture is shown during this chapter. It focuses on the lowest layer of the system –the data layer. Firstly system components are presented, after that the services provided by the data layer are explained.

5.1 System Architecture Overview

There are three components in this system:

5.1.1 Ant-hill Global Lookup Server

A Global Lookup Server –GLS– is a modified Chimera's node whose functions are:

- Manage Chimera's node.
- Route Chimera's messages.
- Store information about each block location taken from the publish messages.
- Reply to lookup messages with information about the block locations when it is available.
- Forward store messages received from Chimera's network to its Local Lookup Servers.
- Listen to its Local Lookup Servers' messages, send them to Chimera's network and reply.

Messages sent over Chimera's network have the format detailed in Appendix A. Chimera's default port is 26580 which will be customisable in future versions, when command options are implemented 9.3.2.

It stores information about networks. It makes the list lighter

This is a link between local networks and the global network –i.e. the Internet.

Listen to LLSs in port 26581.

Information about some subnetworks. It could accept to manage 'homeless' clients connected through a Virtual LLS.

5.1.2 Ant-hill Local Lookup Server

A Local Lookup Server –LLS– is a tracker which manage its client's blocks. Although it is inspired on BitTorrent's tracker, a LLS does not know anything about the semantic layer. So it does not track files anymore, it tracks opaque blocks.

- Store information about the availability of data blocks within the local network taken from the (un)publish messages.
- Reply lookup messages received from its clients and other LLSs. The response is a list of clients which store the data requested.
- Forward lookup requests received from its clients to its GLS when the data is not available on its local network.

A LLS should accept solely clients within the same local network.

It has a caching policy in order to cache blocks according to some rules. For example, it caches every block which is requested by three or more clients.

Virtual Local Lookup Server

It is not an actual component, it is only a light version of a LLS implemented in the Ant client. However, it pretends to be, and is treated as, a real LLS. Each Virtual LLS has to find its closest GLS and GLSs have to accept external LLS joins –at least the LLSs with low latency.

Since it is highly probable that the client's ISP does not install a LLS on its local network, moreover in the earlier age of the protocol's introduction a client could be a 'homeless client'. It that case, it cannot join any LLS (remember that LLSs should only accept joins from its local network's users) and there are two options:

- to install a LLS on the home network
  It is a nice solution for home networks.
- to use a Virtual LLS
  The best solution for a single computer.

Virtual LLS could be also used by users who want to avoid its ISP policies. It is useful in order to detect –and bypass– ISP's cheating behaviour. In that case, it will join a close GLS instead of the closest one.

This module has not been implemented in the current prototype.

5.1.3 Ant Client

An Ant Client connects to its ISP's LLS, when available. In any case, it can connect to a Virtual LLS as it was shown above.

Client's API offers the functionality described in Section 3.2.4. The details of the syntax and meaning of each command are described in Appendix B.

A client builds a message within commands and arguments which is sent to its LLS. bt_* functions allow data transference between clients –a LLS also uses these functions in order to cache blocks–. The functions which a client calls are:

- LMsg ***lmsg_create(unsigned char protocol, unsigned long size)**
  Allocate size bytes in payload. Set the code of the protocol.

- void **lmsg_new(LMsg *msg, unsigned short msgID)**
  Create a formated message (msgID, msgLength, #commands). lmsg_create must be called before to allocate memory enough.
- void **lmsg_add_com(LMsg *msg, unsigned short command)**
  Add a command to the payload. Set # arguments of this command to 0. lmsg_new must be called before the first call of this function.
- void **lmsg_add_*(LMsg *msg, ArgType *arg)**
  Add an argument of the last command added to the payload. lmsg_add_com must be called before. See code's documentation for details.
- unsigned **long payload_to_lmsg(LMsg **msg, char *payload, unsigned long offset, unsigned long length)**
  Where *payload* is a piece of the message received, *offset* is the offset of the *payload* within the message, *length* is the length of *payload*. Return the length of the message created –i.e. offset+length– or 0 when an error occurs. *msg* gets the message created.
- unsigned short **lmsg_num_com(LMsg *msg)**
  Return number of commands of *msg*. payload_to_lmsg or lmsg_receive must be called before.
- unsigned char **lmsg_read_com(LMsg *msg)**
  Return the code of next command. payload_to_lmsg or lmsg_receive must be called before.
- unsigned short **lmsg_num_arg(LMsg *msg)**
  Return the number of arguments of current command. lmsg_read_com must be called before.
- ArgType **lmsg_read_*(LMsg *msg)**
  Return next argument of current command. lmsg_read_com must be called before. See code's documentation for details.
- int **lmsg_send(LMsg *msg, int socket)**
  Send a message through socket. Return -1 when an error occurs, 0 otherwise. See lmsg_send_to_* functions in code's documentation.
- LMsg ***lmsg_receive(int socket)**
  Receive a message through socket and transforms the payload in a LMsg struct. Return the LMsg struct or NULL on error.
- **lmsg_*()** more auxiliary functions.
- void **bt_download_block(Key *block, unsigned long offset, unsigned long length, HostRef *source)**
  Download *block* from *source*. Save the file in blockID.block
- void **bt_start_server(int port)**
  Inicialize the data transfer server on this machine at port. Receive DOWNLOAD messages from ANT Clients and transfer data to them.

5.2 Lookup Service

The final design of the lookup service consists of a network of networks. Each network has to have a Global Lookup Server (a modified Chimera node), a Local Lookup Server (inspired on BitTorrent tracker) and, optionally, a cache. All of these servers should be at the gateway of a ISP's network, however they could run on a personal computer elsewhere.

Clients are connected to a Local Lookup Server which manages their requests. This has a data base with the location of every block available on the local network and replies every local lookup request with a list of clients that have the block requested. It is like a BitTorrent tracker, however, it does not track files; actually, it just tracks the blocks that its users have.

Global Lookup Servers belong to a Chimera network, they manage the information about which blocks are on the whole global network. Of course, a single node has not information about the whole global network, this is distributed along all the nodes on the network. Chimera makes it possible and allows nodes to know in which networks a requested block is.

Figures 5.1 to 5.3 show an example of Global Lookup Service where blockID 3220 is published by one node, it is requested and downloaded by other nodes. The sequence is explained in each figure's caption. See Figures 8.1 to 8.3 to compare with the original behaviour of Chimera/Tapestry.

A detailed view of the lookup service and messaging exchange is available in Appendix B.

5.3 Storage Service

There are some interesting storage features which have been included in this system. They offer new features which could be used in many applications. One of the most important of this system is to pre-fetch some contents before the official publication. Structured lookup protocols distribute the blocks along the nodes that belong the network as is described in Section 8.4.

5.4 Data Transfer

Clients transfer data to each other. After finding the client which actually has a block, the download can start. There are transfers between clients, however, Local Lookup Servers can send and receive data in order to cache data.

A simple custom-made transfer protocol is used in data transfers. The adventages of doing in that way are: it was easy to implement this over the Ant messaging module, it keep homogeneous all the messages involved and it is not necessary an external server. The details of DOWNLOAD and UPLOAD commands are described in detail in Appendix B.

The current implementation is too simple and it may not be useful in a real environment. It only permits downloads but it does not face the negotiation problem. However, a BitTorrent-like protocol seems to be easily implementable using command options, which will be implemented in future versions (see Section 9.3.2).

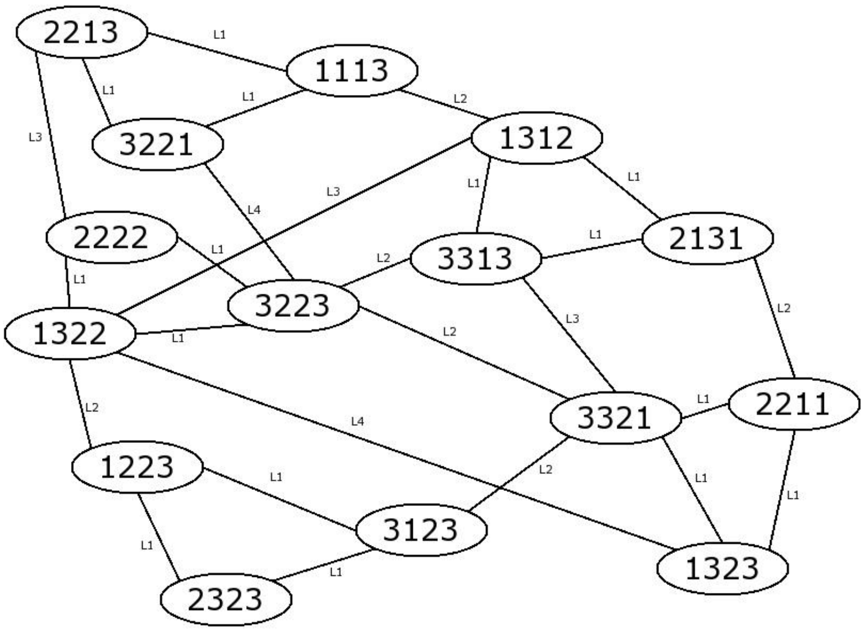

Figure 5.1: Initial state. Some links have been omitted for clarity.

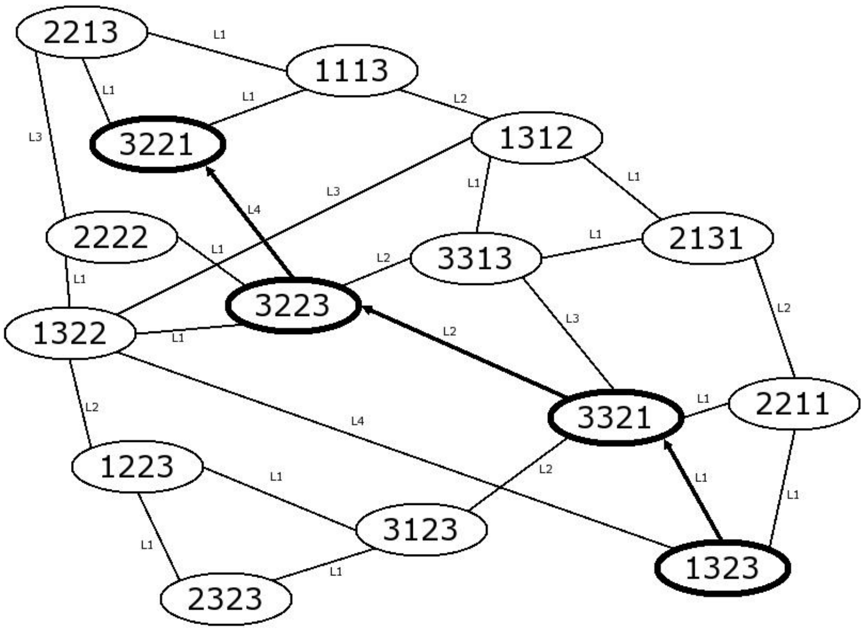

Figure 5.2: A LLS publishes on its GLS –node 1323– blockID 3220. Node 1323 sends the publish message to the Chimera network. Every node in the path stores a reference to the publisher –the LLS–.

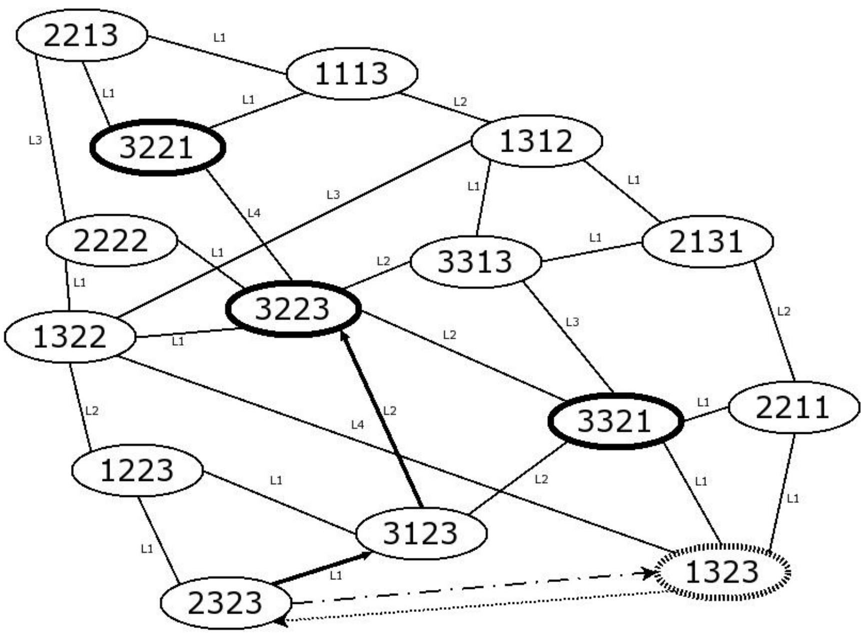

Figure 5.3: Node 2323 requests the block. The message is routed until the closest node which has a reference. Node 3223 replies with the IP address of the LLS that published on node 1323. A client of node 2323 downloads the data from a client of node 1323.

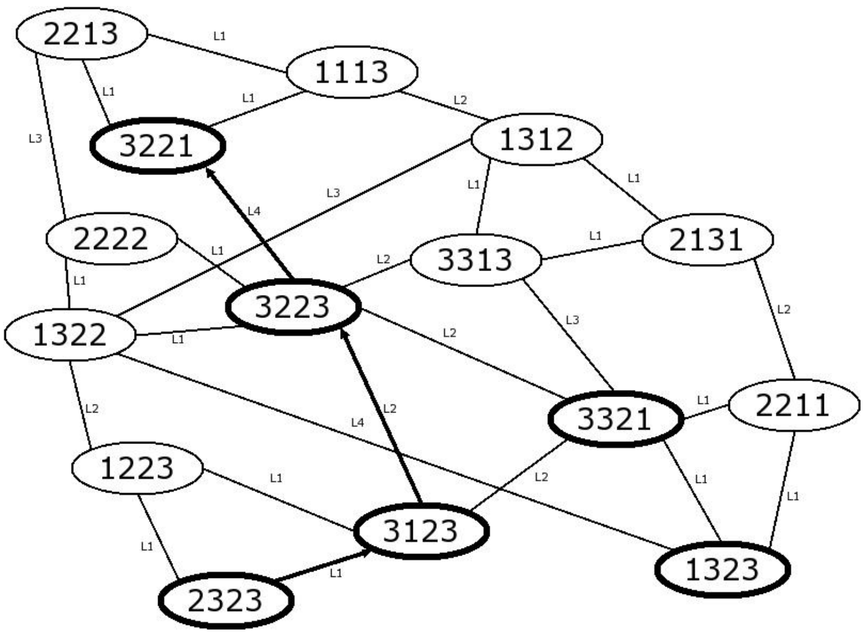

Figure 5.4: Once the download ends. The client publishes the block through its LLS. Node 2323 publish the reference on the Chimera network.

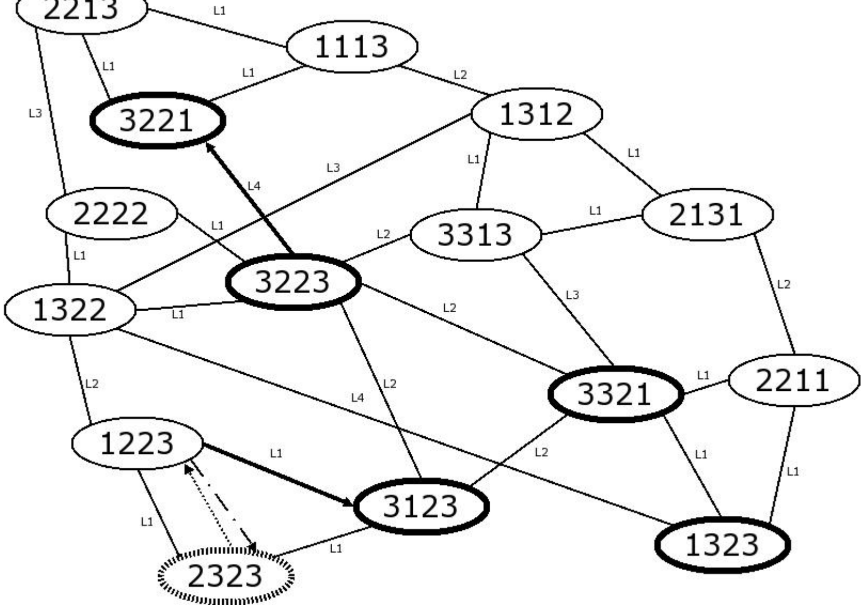

Figure 5.5: A client of node 1223 requests the block. The reply is obtained from the first node in the path, which has a reference to a LLS of node 2323. The client downloads the data from a client of node 2323.

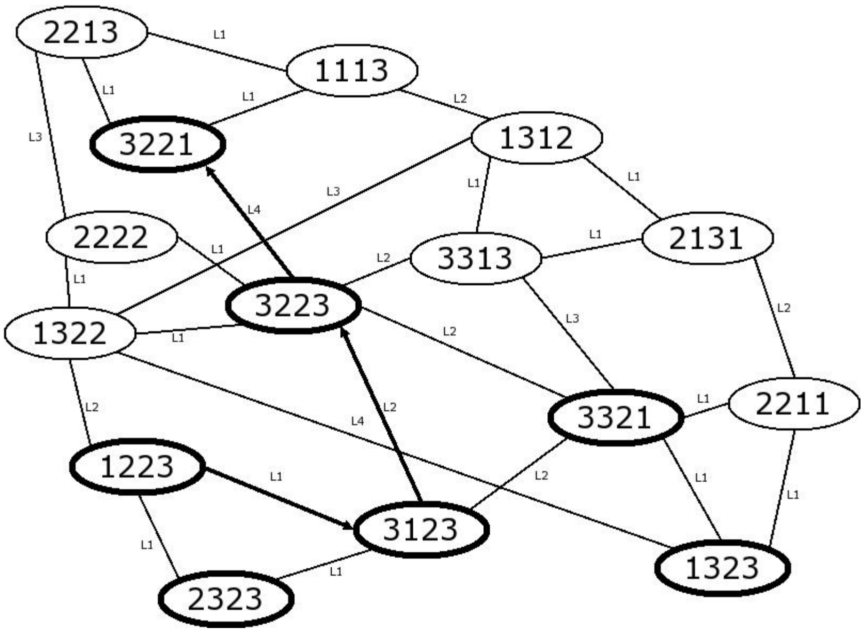

Figure 5.6: Node 1223 publishes blockID 3220. Notice that the publish message is always routed until the root. Hence, node 3123 has two references; nodes 3223 and 3221 have three. Root node stores every reference on the network.

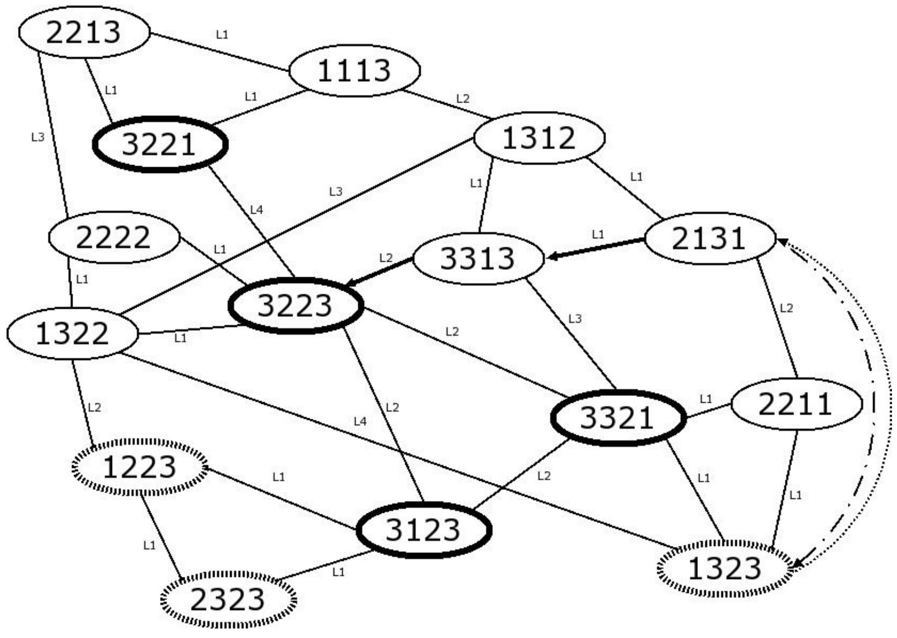

Figure 5.7: When node 2131 requests the block it receives a reply with three references. The LLS can choose which one is the best option –latency, economic, etc–. In this case, the closest host is chosen.

Chapter 6

System Implementation: Ant Prototype

This chapter shows which parts of the system have been implemented. It provides an overview of the prototype; implementation details are not shown here. The complete documentation of the prototype is in *ANT Documentation*.[1]

6.1 Shared modules

The modules listed below are used by two or more components of the system.

6.1.1 blockref

This module provides data structures and functions needed to build a data base within information about which nodes store the data related to each block. In fact, this module could be substituted by a data base in the future.

6.1.2 localmsg

Localmsg module provides data structures and functions to build messages that will be sent and to extract information from received ones.

It is able to build messages up to $2^{32} - 1$ bytes, up to 255 commands within up to 65535 arguments each. Arguments are up to 255 bytes long (except *binary data* argument which could be up to $2^{32} - 1$ bytes long –i.e. 4 GB–). Details about message format are in Section A.1.

Despite this is a general-purpose module, there are specific functions that code/decode specific data structures used by servers and clients.

6.1.3 blocktransfer

Blocktransfer module provides data structures and functions to make it possible data transfers between peers. This is used by clients once the location of the requested block is

[1] http://www.tslab.ssvl.kth.se/users/raul/ant/doc/code/

found. Local Lookup Servers also can use this module in order to store and deliver cached data.

6.1.4 hostref

This module provides an easy way to manage host and lists of hosts. It was designed in order to allow IPv6 addresses, however the IPv6 part has not been implemented yet.

6.2 Ant-hill Global Lookup Server

As it has been explained, a Global Lookup Server –GLS– is formed by a modified Chimera node and a server which listens to Local Lookup Servers' requests.

6.2.1 Modified Chimera Node

In Section 8.6.1, the Tapestry overlay is shown in depth. The Ant system takes advantage of the routing system which Chimera provides, however a GLS replies with a list of LLS where the block is stored instead of the data itself.

A GLS has a table of networks where blocks are stored. This table has information about every block stored within this GLS's LLSs local networks. It has also information provided by Chimera overlay (i.e. the whole list of local networks when it is the block's root and the closest GLSs otherwise).

Globalmsg –a modified version of *localmsg* (see Section 6.1.2)– is used to build messages which will be sent over Chimera overlay. Since Chimera routes messages depending on the block requested, each message only can carry one command for that specific block.

6.2.2 Global Network Gateway

A Global Lookup Server manages requests from its Local Lookup Server(s). Usually each GLS has one or more "official" LLSs within the same network (i.e. one GLS per network and several subnetworks with one LLS each one). However, it should allow requests from close LLSs that are not associated with any GLS.

This module decodes each LLS' message, build a suitable Global request, sends it to the global network, receives the reply, builds a reply message and sends it to the LLS. This is done calling *localmsg*'s and *globalmsg*'s functions.

6.3 Ant-hill Local Lookup Server

6.3.1 Global Lookup Server's Client

LLS sends requests to GLS when it is necessary (i.e. the first publication, the last unpublication and lookups which did not solve locally). Current implementation only takes one command per message, but next versions will be able to collect many requests –from many clients– and send them into a single message. *localmsg*'s functions are called to code messages and decode replies.

6.3.2 Block Data Base

Each LLS stores a data base where are the list of host which store each block within the local network. *Blockref* module is used to manage this data base.

6.3.3 Local Lookup

LLSs listen to its clients, decodes requests and lookup in its local data base. When the information is available on the local network, it replies directly.

6.3.4 Cache

This version does not implement cache feature. In order to provide that, the LLS should use *blocktrasfer* in order to start a service that listens to download requests and uploads data. Moreover, it should set a policy for caching –e.g. cache blocks requested by at least 10 clients–.

6.4 Ant Client

An Ant client provides an interface which allows user to publish, un-publish, lookup and store blocks through its LLS by sending messages built calling *localmsg*'s functions. Apart of that, each client starts a service that listens to download requests and uploads data. A client downloads data calling the proper *blocktransfer*'s function.

Chapter 7

Experiments

Some experiments have been carried out in order to test and debug the prototype. However, some experiments are proposed here to determine the limitations of the system.

7.1 Environment

In this section describes the tools used in the experiments.

7.1.1 PlanetLab

> "PlanetLab currently consists of 694 machines, hosted by 335 sites, spanning over 25 countries."[1]

PlanetLab offers a homogeneous Linux-based environment where distributed systems are tested and measured. That means a binary file compiled on one host will run on the others.

The tests and measurements of this prototype have been performed along over 300 hosts around the world.

7.1.2 Scripts

Some scripts have been written in order to automatize tests. They compile the source code on a host, spread the binaries, execute them and get logs. They are also included in *ANT Documentation*.

> WARNING
> Running these scripts would be consider as an attack against network resources. Contact your network administrator before using them.

7.1.3 Logs

Each GLS and LLS generate logs which show the message exchange. Non-interactive versions of Ant client have been developed in order to automatize publication and lookup requests which also generate logs. Some scripts were also developed in order to get logs from every node, store them and merge them in a single file sorted by time. Some simple

[1]http://www.planet-lab.org/php/overview.php

programs will be able to analyze logs to trace messages, measure latencies, etc. In fact, a program has already written to trace publication messages.

7.2 Tests

Several tests have been carried out on PlanetLab to assure the accuracy of the prototype developed. These allowed to fix some bugs which were not detected on the single-host tests.

It is easy to check a few publications and lookups tracing their paths, although it takes time. As soon as the number of messages grows, an analyzer is needed.

It is possible to perform this kind of experiments in an environment where you have every nodes under control and you receive every log within the network. It will become harder when independent nodes manages the Ant system.

7.3 Measurements

The goal of the experiments described in this section are to demonstrate the feasibility of such a system as well as detecting scalability shortcomings.

The experiments should be run over as much hosts as possible in order to measure topology awareness. It could be also possible use virtual Chimera nodes, which are described in [14]. For example, in a network of 300 hosts, a message will not visit more than four nodes, what causes long distance hops.

7.3.1 Lookup System

Since P2P systems allow clients to transfer data from each other, it provides scalability for this issue. However, the scalability issue is moved to the lookup service which is responsible of manage information about where the data is.

Total Latency

Latency latency is the time between the moment a client requests a block and the moment it receives the reply.

It depends on the availability of the data within the local network.

- A fast response –i.e. RTT[2] + request's processing time in Local Lookup Server– will be obtained when the block data is available on the local network.
- A non-predictable delay will be obtained when the block data is not available on the local network. In this case, Chimera overlay will try to locate the closest Global Lookup Server whose network stores the data. The total delay will be: RTT Client-LLS + request's processing time in LLS + RTT among Chimera nodes in the path + request's processing time in each GLS + RTT to the closest LLS which has the data + request's processing time in this LLS + reply's processing.
- A even less predictable delay will be obtained when the block requested does not exist in the whole global network. Then, Chimera overlay will route the request until the blockID's root whose location is totally arbitrary.

[2] Round-Trip delay Time

Latency Added By Local Lookup Server

Once a Local Lookup Server receives a request, it looks up it in its own data base. If there are references, they will be put into a reply message, otherwise a lookup message will be built and sent to its Global Lookup Server.

These operations should be measured in order to improve the speed and reduce memory consumption. They could cause a bottleneck in the local lookup service. It could affect the whole lookup service by locking LLSs which try to locate a local host storing a block given.

Latency Added By Each Global Lookup Server on the Path

Once a Global Lookup Server receives a request, it builds a suitable message and sends it to Chimera network. Since this server is the first node in the path, its forward –or delivery– call-back function will be executed. Both functions look up in the node's data base; if there are references, a reply message will be built and sent back, otherwise forward function will route the request to the next node in the path and delivery function will reply a empty list as result.

These operations should be measured in order to improve the speed and reduce memory consumption. They could cause a bottleneck in the global lookup service –i.e. the scalability of the system depends on the performance of these operations–.

Notice here security counter should be implemented in order to circumvent nodes which provide false data or introduce excessive latency. However, these issues are out of the scope of this thesis.

System Requirements and Limitations

Since *blockref*, *localmsg*, *globalmsg* allocate dynamic memory an excessive amount of memory could produce a denial of service. It should be measured the amount of memory required in publication, the residual memory after unpublish command and the amount of memory required by messages arriving and departing. Since each server can manage several connections at once, the amount of connections and messages should be measured as well as limits which count denial of service fails.

7.3.2 Identifying Bottlenecks

The strongest limit of a data distribution system is widthband used to transfer data. This limit should be the limit of the system, hence the lookup service has to be able to manage the amount of requests generated by clients which are in the limit.

The requirements of the lookup service could be calculated in function of the data each client can transfer plus the lookup operations demanded by other GLSs and LLSs.

The maximum number of requests generated by local clients depends on: amount of data they request, block size and the amount of clients.

requests = (data requested/block size) number of clients*

7.3.3 Measurements To Counter Bottlenecks

Once the bottlenecks are identified, specific measurements should be performed in order to analyze their cause. Then, some modifications should be designed and implemented.

The former measurements will be performed again to compare and decide whether the modification improves the system or not.

Chapter 8

Related Work

In this chapter, the related work is presented. In Chapter 4 is written the research process.

8.1 Definitions

peer-to-peer There is not a single definition of peer-to-peer (P2P). We found some of them along the papers we had read. It is HP's definition written in [10].

> "The term **peer-to-peer** refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing, data/content sharing, communication and collaboration, or platform services."

peer A computer or any kind of machine which is connected to a P2P network. It should have a lookup mechanism. And it will be able to offer blocks and download blocks from other peers.

lookup A function used by peers in order to locate a block.

block A piece of information. Usually, files that are being shared are divided on several blocks.

scalability Ability to work without service degradation regardless number of users and files on the network.

8.2 The peer-to-peer Model

8.2.1 History of P2P networks

The structure of this section is taken from [1].

Napster: the beginning of a new age

Napster [1] was the first widely used P2P network. Since its publication in 1999, P2P has been a hot topic in the Internet community. The number of users of P2P networks grows every day regardless legal and technical troubles.

The Napster protocol is as simple as popular. Users share their files by publishing them on a central server. They also ask the central server for file they want and the central server responds with the list of users who are sharing the file requested. Finally, users download files from each others. The server is only a directory server, it does not store the files itself.

There are several problems in that protocol. Firstly, the central server is a single point of failure. Whether it is affected by a legal and technical threats. In fact, in July 2001, it was shut down by a court order [2] due to most users used to share contents under copyright protection. Its second problem is about scalability. Since the load of the central server grows as fast as number of users, it cannot manage an unlimited number of them.

Evolution and spreading of file sharing systems

After Napster shut down, many developers tried to build better protocols. Some of them, such as Gnutella [3] and Freenet [3], do not use any kind of central server. They implement "pure P2P" systems where peers are equal each other. The main problem of these implementations is caused by their lookup technique. They flood the network by asking neighbors and these nodes do the same until the block is found. This schema is very useful in order to achieve anonymity but it is not efficient on large networks –i.e. it is not scalable–.

An optimization to the flooding approach was introduced by KaZaA and latest versions of Gnutella uses it as well. In these systems there are a set of super-peers which act as directory servers in order to reduce the amount of flooding produced by queries.

Nowadays, according to a research by CacheLogic[4] [2], the most used protocols are BitTorrent and eMule.

8.2.2 BitTorrent

BitTorrent[5] is a P2P file sharing protocol[6] where each file is managed by a tracker[7]. A tracker is a server that manages information about peers which are sharing a file. Someone who wants to publish a file should find a tracker and make a torrent file. This file contains information about the shared file and the tracker that controls its distribution. Then, another user who wants to download that file will use the torrent file to connect the right tracker.

This schema has proved to be very effective although it relies on a tracker –or a limited set of trackers– and it is not locally aware. Despite that, this protocol is used by millions and it generates the third part of total traffic on the Internet according to [2].

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

[2]http://en.wikipedia.org/wiki/Napster

[3]Gnutella, http://genutella.wego.com

[4]Cache Logic is a company with commercial interest in peer-to-peer caching. Its conclusions should be taken carefully.

[5]BitTorrent, http://en.wikipedia.org/wiki/Bittorrent

[6]BitTorrent's Specification, http://www.bittorrent.com/protocol.html

[7]Actually, a file could be managed by several trackers. That is a simplification.

Some technical details of the BitTorrent protocol are published on its official web site [8].

8.3 Commercial Distributed Delivery Systems

Despite some content providers starts to offer digital content over P2P systems, there is few information about how these systems work. In this Section is shown a commercial distributed delivery product.

8.3.1 Kontiki Delivery Management System

As it is claimed in their own papers[9]:

> "The Kontiki Delivery Grid is a secure, centrally managed, high-performance, enterprise-ready implementation of grid delivery for legal distribution of legitimate content to authorized users."

This system is focused on the control of users, publishers and contents. The system relies on secure connections over HTTPS and DRM –Digital Rights Management– to control everything within its Kontiki network.

The core of the system is formed by a set of servers which store the data performing replication on different locations in order to make it failure-resilient and to store replicas near consumers. The network is centrally controlled and every user has to be registered in a LDAP-like directory. That means each user has restricted access to some material and only the users who have permission can publish on the network.

Publishers can give clients a link on a web page, then the user follows the link and the system authenticates him through user's name and password and checks user's permissions over the file requested; if the client is allowed to download the file, it will download it from servers and from other close users.

The system inherits scalability and reliability from P2P model. It also offers a feature called "Time Shifting" which faces peaks by downloading data at off-peaks hours.

This system is conceived as an enterprise application for a large company's network where all the content is controlled by the company. Publication is restricted and there is a exhaustive control over the distribution of contents –who watch what content at that time on which computer–. It is possible due to a centralized management of the grid and DRM applications on the client side. That is useful for companies which want to have access to confidential content at any location –specially useful when the content is large such as a multimedia file–.

It offers a secure system, prevents users to access confidential data or prevents users to copy and forward through DRM applications such as Windows Media Player for multimedia files and Adobe Secure Document Control for Adobe PDF files. Unfortunately the perfect security does not exist and even using DRM is very difficult to prevent the copy of confidential material. It is trivial case where anyone, who is not allowed to copy or print a document, can take a picture of a DRM-protected document that is shown on a computer screen. That means it is a good system but none company can say "X system gives us complete security, we do not have to do anything else".

[8]http://www.bittorrent.org/protocol.html

[9]Kontiki, http://www.kontiki.com/technology/index.html

Some digital content distributors use Kontiki to distribute their movies as Kontiki's web page claims[10]:

> "TimeWarner's AOL/Moviefone is using Kontiki to deliver DVD quality movie trailers to Internet users." "Cinequest provides DVD quality independent films and trailers to millions on Internet users."

BBC also developed its own distribution system –integrated Media Player (iMP)[11]– over Kontiki.

8.4 Structured Lookup Protocols

In response to scalability problems of peer-to-peer file sharing systems, some groups have propose several structured lookup protocols based in Distributed Hash Tables [11]. In these protocols, each block of information is identified with a key –its blockID–. This key is the result a hash operation of the data of the block and –optionally– some other information such as the size of the block, a salt [14],etc. In the same way, nodes have a unique long key –by hashing their IP addresses, for example–.

In [7] are shown the properties that are desirable from a location-independent routing infrastructure. The list below is an extract from [7]:

1. *Deterministic Location*: Objects should be located if they exist anywhere in the network.
2. *Routing Locality*: Routes should have low stretch[12], not just a small number of application-level hops.Sending queries to the nearest copy across the shortest path possible is the ideal.
3. *Minimality and Load Balance*: The infrastructure must not place undue stress on any on its components; this implies minimal storage and balanced computational load.
4. *Dynamic Membership*: The system must adapt to arriving and departing nodes while maintaining the above properties.

The main difference between this kind of protocols is their routing algorithms whose characteristics are shown along this section. Despite that, all of them offer scalability and allow to locate objects in, at most, $O(logn)$ hops.

They offer the same functionality. In fact, there is "an ongoing effort to define common APIs for structured peer-to-peer overlays" described in [4]. In fact, this section is based on [4][13]. It makes an abstraction of lookup protocols into three layers:

- Tier 0: Key-based Routing Layer (KBR)
  KBR is a virtual tier which forms the common denominator of services provided by existing structured overlays
- Tier 1: Structured Overlays
  Lookup Protocols: DHT and DOLR

[10]http://www.kontiki.com/customers/

[11]BBC's iMP: http://www.bbc.co.uk/imp/

[12]"Stretch is the ratio between the distance traveled by a query to an object and the minimal distance from the query origin to the object" (definition from [7])

[13]CAST and I3 are not mentioned in this paper due to they are out of the scope of this thesis.

- Tier 2: Applications
  CFS, PAST, I3, Scribe SplitStream, Bayeux and OceanStore are known applications over structured overlays. The Ant protocol belongs to this tier.

A node is the name of any participant in the overlay. It is identified by its nodeID and is reachable through an IP address:port pair. Each nodeID is a pseudo-random large number (at least 160 bits).

Each key has a node which is responsible for it, this node is the key's root. Each node maintains a a routing table which stores a set of pairs (nodeID, IP address:port). Messages, whose destination is a key, are routed by nodes to a closer nodeID. When a node's nodeID matches the message's key –or the node does not know another nodeID closer to the key– the message has reached its key's root.

Each system defines a function that maps keys to nodes.

8.4.1 Tier0: Key-based Routing API

This section is an extract from [4].

Data Types

A *key* is a 160-bit string. A *nodehandle* encapsulates de transport address and nodeId of a node in the system. The nodeId is of type *key*; the transport address might be, for example, an IP address and port. Messages (type *msg* contain application data of arbitrary length.

We adopt a language-neutral notation for describing the API. A parameter p will be denoted as ->p if it is a read-only parameter an <->p if it is a read-write parameter. We denote an ordered set p of objects of type T as T[] p.

Routing Messages

- void **route(key ->K, msg ->M, nodehandle ->hint)**
  "This operation forwards a message, M, towards the root of key K. The optional hint argument specifies a node that should be used as a first hop in routing the message. A good hint, e.g. one that refers to the key's current root, can result in the message being delivered in one hop; a bad hint adds at most one extra hop to the route. Either K or hint may be NULL, but not both. The operation provides a best-effort service: the message may be lost, duplicated, corrupted, or delayed indefinitely.

  The **route** operation delivers a message to the key's root. Applications process messages by executing code in upcalls which are invoked by the KBR routing system at nodes along a message's path and at its root. To permit event-driven implementations, upcall handlers must not block and should not perform long-running computations."

- void **forward(key <->K, msg <->M, nodehandle <->nextHopNode)**
  "This upcall is invoked at each node that forwards message M, including the source node, and the key's root node (before deliver is invoked). The upcall informs the application that message M with key K is about to be forwarded to nextHopNode. The application may modify the M, K, or nextHopNode parameters or terminate the message by setting nextHopNode to NULL.

  By modifying the nextHopNode argument the application can effectively override the default routing behavior."

- **void deliver(key ->K, msg ->M)**
  "This function is invoked on the the node that is the root for key K upon the arrival of message M. The **deliver** upcall is provided as a convenience for applications."

Routing State Access

The API also offers other functions which access a node's routing state. All of these operations are strictly local and involve no communication with other nodes.

Since their specification is too detailed for the porpoise of this paper, only **update** upcall and **range** function are explained and the rest are listed. For further details see [4].

- nodehandle[] **local_lookup(key ->K, int ->num, boolean ->safe)**
- nodehandle [] **neighborSet (int ->num)**
- nodehandle [] **replicaSet (key ->k, int max ->rank)**
- void **update(nodehandle ->n, bool ->joined)**
  "This upcall is invoked to inform the application that node has either joined or left the neighbor set of the local node as that set would be returned by the neighborSet call."
- boolean **range (nodehandle ->N, rank ->r, key <->lkey, key <-rkey)**
  "This operation provides information about ranges of keys for which the node is currently a r-root. The operations returns *false* if the range could not be determined, *true* otherwise. It is an error to query the range of a node not present in the neighbor set as returned by the **update** upcall or the **neighborSet** call. Certain implementations may return an error if is greater than zero. $[lkey, rkey]$ denotes an inclusive range of key values.

  Some protocols may have multiple, disjoint ranges of keys for which a given node is responsible. The parameter lkey allows the caller to specify which region should be returned. If the node referenced by N is responsible for key lkey, then the resulting range includes lkey. Otherwise, the result is the nearest range clockwise from $lkey$ for which N is responsible."

  This function is not implemented by Chimera. It is necessary in order to determinate which range of published keys has to be sent to a joined node.

8.4.2 Tier 1: Lookup Protocols Abstraction

This section is a schematic summary of [4].

- The *DHT abstraction* provides the same functionality as *a traditional hash table* –i.e. put(key,data), remove(key) and get(key)–. In fact, it maps each key into a node which stores the data given by put function. These data can be objects of any type. For example, CFS stores and retrieves single disk blocks by their content-hashed keys.
- The *DOLR abstraction* provides *a decentralized directory service* and it provides three functions: publish(objectId), unpublish(objectId) and sendToObj(msg, objectId, [n]).

  As opposed to DHT, DOLR permits to store replicas anywhere within the system. Nodes notify the (un)availability of an object by sending a (un)publish message to the objectId's root. Every node in the path stores the (node, objectId) pair. Each message sent by calling sendToObj(msg, objectId, [n]) is forwarded by nodes until it arrives in a node which stored a (node, objectId) pair. Then, this node routes the message to the publisher node providing a locality property not supported by DHTs.

8.4.3 Tier 2: Applications

CFS, PAST, I3, Scribe SplitStream, Bayeux and OceanStore are known applications over structured overlays. The Ant protocol belongs to this tier.

8.5 Chord

Chord [12] is a DHT protocol which maps keys to hosts. Every hosts in the network form a ring where each host is responsible of the keys between its predecessor's nodeID and its own nodeID. It is only necessary a link to successor in order to provide a lookup system.

To improve lookups each node stores a list of hosts –fingers– which assures, at most, log(h) hops per lookup query. In fact, nodes can store a list of cached nodes which could reach the destination in a single hop. It means that the path of a query is not static; in opposite of Tapestry which always routes messages in the same way without taking shortcuts.

However, successors and predecessors are arbitrarily near what causes arbitrary large hops even when the destination is topologically close.

8.6 Chimera: An Implementation of Tapestry

Chimera is light-weight implementation of Tapestry in C. It is under development, that means some functions have not been implemented yet and there are some parameters which are currently ignored. Current version is 1.04

Chimera follows the syntax defined above. Actually, it does not offer publish or unpublish functions. However, a publish function is emulated by building a publication message and sending it. It is more flexible than the original publish function due to the application layer is able to modify Tapestry behavior. It is important for this system since it needs a modified behavior of lookup which gets a list of host instead of the data.

The main function of Chimera API is **void chimera_send(ChimeraState *state, Key key, int type, int len, char *data)** which is the equivalent to **route**.

It also offers three up-call functions: **chimera_update**, **chimera_forward** and **chimera_deliver** that are equivalent to the Key-based Routing functions.

Regarding to the routing interface, it offers the following functions: **ChimeraHost **route_lookup(ChimeraState* state, Key key, int count, int is_safe)**, **void chimera_route(ChimeraState* state, Key *key, Message *msg, ChimeraHost *hint)** and **ChimeraHost **route_neighbors(ChimeraState* state, int count)**. There is not a function equivalent to **range**

- void chimera_send(ChimeraState *state, Key key, int type, int len, char *data)
- ChimeraHost *host_get(ChimeraState *state, char *hn, int port)
- void host_release(ChimeraState *state, ChimeraHost *host)

8.6.1 Tapestry

Once the API has been shown, a deeper description Tapestry is given in this section.

The definition of Tapestry is given in [13]:

> "Tapestry is a peer-to-peer (P2P) overlay network that provides high-performance, scalable, and location-independent routing of messages to close-by endpoints, using only localized resources. The focus on routing brings with it a desire for efficiency: minimizing message latency and maximizing message throughput. Thus, for instance, Tapestry exploits locality in routing messages to mobile endpoints such as object replicas; this behavior is in contrast to other structured P2P overlay networks[14]."

As it was written before, each node has a route table. A Tapestry node maintains a table, called *neighbor maps*, which is sorted according to the matching prefix between this node and its neighbor's key. It is sorted by levels (e.g. a node's key $6E83$: neighbors whose key is $6E8*$ is a level 4 neighbor, level 3 when $6E**$, level 2 when $6***$ and level 0 when the first digit does not match). This mechanism is similar to CIDR IP routing system [15].

In Figure 8.1 some nodes are linked by levels 1 to 4 links. A message is generated in node 5230 whose destination is 42AD. The source node looks in its routing table for the longer prefix that matches –i.e. the greatest level link–. In this case, a level 1 link is chosen. Each node performs the same operation, taking each hop a greater level link. This is the reason for what a message reach the destination in, at most, $log_\beta(N)$ hops; where IDs are in base β –in this example IDs are 16-bit long and β is 4–. Since in each hop the closest node is chosen, Tapestry is also locally aware.

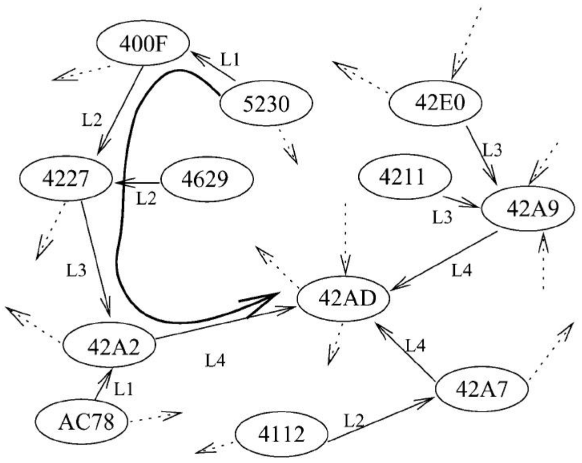

Figure 8.1: *Path of message.* The figure shows the path taken by a message whose source is 5230 and its destination 42AD. (source: [13])

In Figure 8.2 nodes 4228 and AA93 publish they store document 4378. Messages are routed to document's root (4377 in this case); in addition, each node in the path stores a

[14]CAN, Pastry, Chord, Kademlia, Viceroy and Skipnet

[15]CIDR specification on RFCs 1518 and 1519. See also http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

(document,node) pair. If we see the document's tree, we can notice that the root has a complete list of nodes which store the document and nodes in each path –branch– have the list of publications in its paths –branches–.

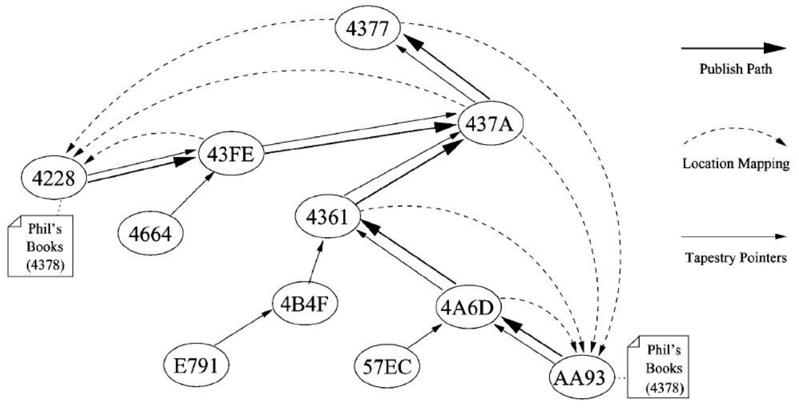

Figure 8.2: *Publication in Tapestry.* The figure shows the path taken by two publish messages whose sources are 4228 and AA93. (source: [13])

This system allows to reply a lookup query from any branch which has information about the location of the document instead of taking the location from the document's root. In Figure 8.3 is shown how node 43FE forwards the message to 4228 saving 2 hops. Notice here that the fist hops are closest than the last; because of this it is said that Tapestry is locally aware.

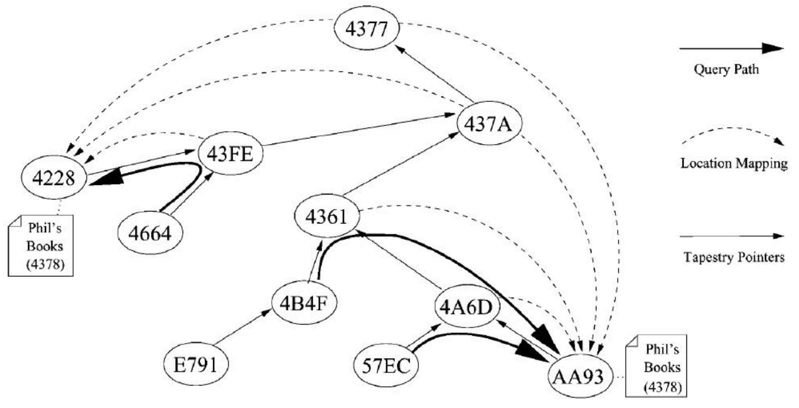

Figure 8.3: *Lookup in Tapestry.* The figure shows the path taken by two messages whose destination is 4378. (source: [13])

Chapter 9

Conclusions & Future Work

9.1 Conclusions

This master thesis has established a guideline to implement a content storage and delivery system that faces most of the problems generated by the distribution of large files over the Internet.

The global design is defined, however, the layers are independent what allows specific modifications without affecting the others.

A prototype of the data layer has been implemented and tested along over 300 nodes around the world. Some measurements should be done to demonstrate the scalability of the system and to find out bottlenecks which treat it.

This is just the beginning, there is a large list of tasks to do. Hopefully, this thesis will help to make better distribution systems or even, who knows, it become a successful protocol over the Internet.

9.2 Lessons Learned

- Papers describe wonderful systems but the real world is cruel.
  When you try to adapt the system to your project, you realize the implementation does not follow specifications or even does not work in your environment. Choosing a overlay becomes more difficult when the documentation is not clear enough.
- It is not trivial to show how your system works.
  Since you are too much familiarised with your system, it is difficult to find the points which should be explained more detailed and the explanations that are too obvious. Benchmarking is even harder to schedule for similar reasons.
- A prototype do not have to implement a final version of the whole protocol.
  Instead of that, it should focus on some specific features which demonstrate the feasibility of the system.
- A prototype should not implement additional features when they add complexity.
  I tried to implement multi-command and multi-argument messages. However, it adds too much complexity; it is not even clear this feature improves efficiency.

9.3 Future Work

9.3.1 Joins and Leaves

Current implementation ignores joins and leaves. This produces undesired effects, the worst of them is that some publications could get lost when the key's root leaves the network. In order to count this issue Chimera's *range* function should be used as soon as it is implemented in Chimera.

9.3.2 Commands Options

Current version of the Ant protocol utilises some fixed parameters as port numbers, maximum number of results of a lookup, etc. The idea of providing command options is to make it more flexible keeping the current message syntax.

This is a list of some interesting options that could be implemented:

- (UN)PUBLISH
  - **port** Publisher's port (when it is not listen to the default port)
  - **only-local** (Un)publish only on the local network.
- LOOKUP
  - **min-results, max-results** Min/maximum number of results requested.
  - **min-hops, max-hops** Min/maximum number of hops (0 means the local/global server should return only results stored on its database).
  - **get-llss** A client could set this option to obtain a list of LLSs instead of a list of clients.
- LOOKUP_REPLY
  - **LLS flag** The results are LLSs instead of clients.
  - **caching flag** The LSS is caching the requested block.
  - **high latency flag** The data is on the network but it could be better to wait until there is a closer replica

In order to allow larger blockID key space, messages between GLSs (over Chimera overlay) must include blockID as an argument –instead of getting that from the Chimera's message–.

9.3.3 Virtual Local Lookup Server

As it was pointed in Section 5.1.2 Virtual LLS is not implemented in Ant. It should be implemented as an extension of an Ant client.

9.3.4 Semantic Layer

This paper presents a guide of how to design the semantic layer. However, the data layer is independent enough to offer flexibility.

Semantic File

A semantic file could take the most part of the fields from BitTorrent meta-info file (.torrent). The fields of a .ant file could be:

- Key of the semantic file.
- Name of the actual file.
- Block length.
- List of blockIDs which form the file.
- Cryptographic key which decrypts the blocks. (optional)
- Information about the publication: author, distributor, date, etc. (optional)
- A set of hosts where there are replicas. (optional)
- Comments. (optional)

9.3.5 Policy Layer

This layer is responsible of caching data, the distribution and the storage. It should offer ISPs flexibility to make they cut down the inter-ISP traffic and make the local traffic more efficient as well.

This layer do not have to –even it should not– know any semantic information about the blocks. In this way the system provides network neutrality. Nevertheless, clients could set some message options in order to indicate whether a block should be cached or not –e.g. a user could indicate “do not cache” when he looks up blocks which belong to a private file shared by a few people–.

9.3.6 Data Layer

The system should allow blockIDs of any length. Current version only manages 160-bit blockIDs due to Chimera also only manage that. However, the system –messages and route tables– are designed to allow keys arbitrary long. A new *key package* should be implemented and a translator which convert any key to a 160-bit key which will be managed by Chimera overlay. This may cause some conflicts when two blockIDs are translated to a single 160-bit key, nevertheless, it does not represent a problem since all components would store the actual blockID instead of the 106-bit version.

Data Transfer

The current version implements a very simple data transfer protocol where a client sends a download message and the receiver replies a upload message with the data. Any client could generate lots of useless messages and make it difficult for the other clients the measurement of their upload/download rate. A BitTorrent-like protocol could be implemented. Even, clients could send the bitmap of the blocks they already have to each other. Although this implies semantic knowledge, those clients already know the semantic information and this feature improves download efficiency. Anyway this feature should be optional.

One of the weak points of BitTorrent is that there are not incentives to make users to seed a file after downloading it. In the Ant system, despite LLS can cache some blocks, users should be encouraged to keep sharing the blocks. This subject should be researched in depth.

9.3.7 Pre-fetching Feature

This feature is called to be the bastion of this protocol due to there is not any peer-to-peer system which offers a way of exploiting the "fashionable gap" (see Introduction).

A regular semantic file could be used to spread the blocks in pre-fetching mode. It would not content the cryptographic key and the list of block could be unsorted.

Moreover, a publisher could publish new contents over RSS channels. Semantic files would content information about several RSS channels –e.g. distributor's, director's, actor's, . . . channels–. Each user would choose which channels he wants to subscribe and each time a feed arrives he will be asked to pre-fetch the content.

For example, a user download a Futurama's episode. At this moment, the user subscribe to Futurama's RSS. As soon as another Futurama's episode is published, the user will be asked to pre-fetch it. Then, the user accepts and downloads the content. At the premier's time, our user gets the semantic file within the cryptographic key –via mail, on a virtual shop, . . . – and watches the episode immediately, at the same time lots of people do. Actually, it simulates TV broadcasting but adding all the digital advantages.

9.3.8 Contextual Advertising

Since P2P file sharing users get digital content for free it seems to be very hard to convince them to download the contents from the Ant system in a legally way. Many of them will get legal if the contents are low price –or even for free–. It could be possible attaching advertisements to the content and on the publisher's web page.

Google's success relies on contextual advertisements –i.e. different ads are shown according to some parameters as a web page's text–. It could be implemented in the Ant system in order to attach dynamic ads to a single content –a movie, for example–. It means that a distribution company could make a profile of each user and show specific advertisements according to his/her preferences, age, location, etc.

Technically, this feature is easily implementable, a movie would be formed by a set of common blocks and a few dynamic blocks. Then, a semantic file would have information about both set of blocks. In order to prevent anyone remove advertisement blocks, each of them has also a part of movie data, causing a jump in the movie when an advertisement block is removed.

9.3.9 Open Future

Since the source code and documentation generated are free –GNU-GPL and GNU-FDL, respectively– anyone is allowed –and encouraged– to improve this system and/or to build his/her own using this material. Of course, developers have to respect the licenses and keep it free.

Anyone is allowed –and encouraged– to make his/her own software compatible with the Ant protocol. In this case, the software could be published under any license.

Anyhow, the author will be glad any suggestion. Please, do not hesitate to contact if you have any doubt or develop anything related to this thesis.

Appendix A

Message format

The syntax of the fields used in Ant messages is:

- <list-of-commands> ::= | <command><list-of-commands>
- <command> ::= <command-code><number-of-arguments><list-of-arguments>
- <command-code> ::= <number>
- <number-of-arguments> ::= <number><number>
- <list-of-arguments> ::= | <argument> | <argument><list-of-arguments>
- <argument> ::= <number><argument-value> | <binary-data>
- <argument-value> ::= | <blockID> | <host-ref> | <long>
- <blockID> ::= <number>*20
- <host-ref> ::= <host-address>
- <host-address> :: <host-ipv4> | <host-ipv6>
- <host-ipv4> ::= <number>*4
- <host-ipv6> ::= <number>*16
- <long> ::= <number>*4
- <number> ::= any decimal integer 0 through 255 (1 byte)
- <end-of-list> ::= 0x00 (1byte)
- <binary-data> ::= | <number><binary-data>

A.1 Local Messages

Local messages are sent and received by Clients, Local Lookup Servers and Global Lookup Servers. They are sent over TCP.

A.1.1 Header

Every local message in ANT Protocol has this header:

- (1 byte) Protocol version
- (2 bytes) Message ID
- (4 bytes) Length of the message
- (1 byte) Number of commands

A.1.2 Commands

This is the format of every command in the version 0.1 of the protocol:

- (1 byte) Command code
- (2 bytes) Number of arguments
- (unfixed) List of arguments

This is the list of commands and their values in decimal:

- L_JOIN 0
- L_LEAVE 1
- L_PUBLISH 2
- L_UNPUBLISH 3
- L_LOOKUP 4
- L_STORAGE_OFFER 5
- L_STORE 6
- L_DOWNLOAD 7
- L_UPLOAD 8
- L_JOIN_REPLY 10
- L_LEAVE_REPLY 11
- L_PUBLISH_REPLY 12
- L_UNPUBLISH_REPLY 13
- L_LOOKUP_REPLY 14
- L_STORAGE_REPLY 15
- L_STORE_REPLY 16

A.1.3 Arguments

- L_PUBLISH, L_UNPUBLISH, L_LOOKUP, L_STORE and L_DOWNLOAD have an unfixed number of arguments whose <argument-value> are <blockID>.
- L_LOOKUP_REPLY has a list of lookup's results. Each result has a former argument whose <argument-value> is <blockID>, an unfixed number of <host-address> arguments and an ending argument (1 byte whose value is 0). For example:

  <L_LOOKUP_REPLY><6><20><blockID1><4><host1><0><20><blockID2><4><host2><4><host3><0>

  It means: there are 6 arguments, the first one is 20-bytes-long (a blockID), the next is 4-bytes-long (the IPv4 address of a host which has information about the previous blockID), the next byte marks the end of list, etc.
- L_STORAGE_OFFER has a single argument whose <argument-value> is <long> (4 bytes) and indicates the size of disk space which this host offers to storage data. It means its maximum value is $2^{32} - 1$ (almost 4 TB.)
- L_UPLOAD always has 3 arguments: blockID, a <long> argument within length of the data (in bytes) and the binary data. An example of a L_UPLOAD command:

  <L_UPLOAD><3><20><blockID><4><1048576><1 MB. of data>

A.2 Global Messages

Global messages are sent and received by Global Lookup Servers over Chimera overlay. It is similar to the local messages, however, each global message can only carry one command. Since each Chimera message is sent to the requested blockID and has a command field, these argument are not present within the Ant message.

A.2.1 Header

Every global message in ANT Protocol has this header:

- (1 byte) Protocol version
- (2 bytes) Message ID
- (4 bytes) Length of the message

A.2.2 Commands

This is the format of every command in the version 0.1 of the protocol:

- (1 byte) Command code
- (2 bytes) Number of arguments
- (unfixed) List of arguments

This is the list of commands and their values in decimal:

- G_PUBLISH 15
- G_UNPUBLISH 16

- G_LOOKUP 17
- G_LOOKUP_REPLY 18
- G_STORE 19

Values under 15 are reserved by Chimera.

A.2.3 Arguments

- G_PUBLISH, G_UNPUBLISH, and G_STORE do not have arguments due the receiver takes blockID from Chimera's layer.
- G_LOOKUP has an argument whose <argument-value> is <blockID>. It is the nodeID of the Global Lookup Server which sent the message –i.e. where the reply will be sent–. The receiver takes blockID from Chimera's layer.
- G_LOOKUP_REPLY has a former argument whose <argument-value> is <blockID>, an unfixed number of <host-address> arguments and an ending argument (1 byte whose value is 0). For example:

  <G_LOOKUP_REPLY><5><20><blockID1><4><host1><4><host2><4><host3><0>

  It means: there are 5 arguments, the first one is 20-bytes-long (a blockID), the three next are 4-bytes-long (the IPv4 address of a host which has information about the previous blockID) and the last byte marks the end of list.

Appendix B

Messaging exchange

B.1 Lookup Service

Figure B.1 shows an overview of a block's publication on a network (messages 1-4), a lookup from another network (messages 5-14) and the download of the acutual data (messages 15-16).

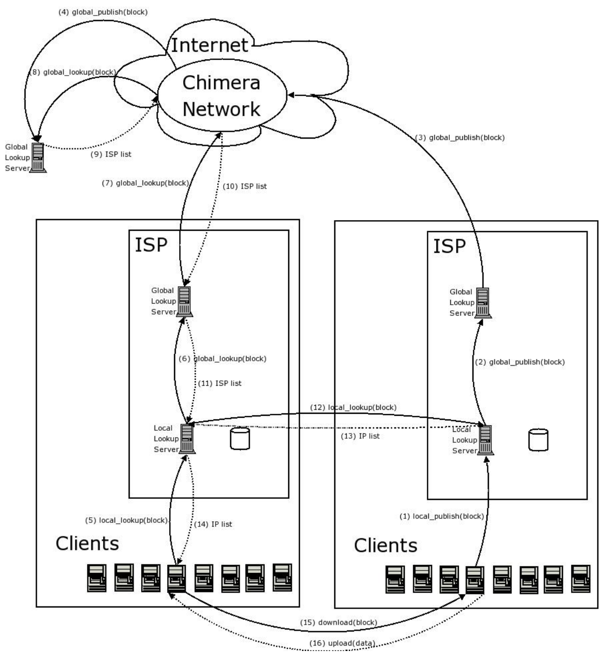

Figure B.1: Overview of Ant messaging exchange.

B.1.1 Client

A client receives commands from the user's interface. They are put in a message that is sent to its Local Lookup Server:

- L_JOIN
  It is sent in the first message to its LLS.
  In future versions it could include some options.
- L_LEAVE
  It is sent in the last message to its LLS. In future versions it could include some options.
- L_PUBLISH blockID
  This client makes the block available for downloading.
  Currently, this command's reply is ignored. In future versions it could include useful information about the publication.
- L_UNPUBLISH blockID
  This client makes the block unavailable for downloading.
  Currently, this command's reply is ignored. In future versions it could include useful information about the publication.
- L_LOOKUP blockID
  The client wants to know the location of the block.
  This message will be replied by a L_LOOKUP_REPLY with a list of clients which have the block, this list will only be empty when the block does not exist on the whole network. In future versions it could include some options such as minimum/maximum number of peers, only local lookup, etc.

B.1.2 Local Lookup Server

A local lookup server receives messages from its clients. These are the commands received and the actions token.

- L_JOIN
  Currently this message is ignored. In future versions it could include some useful information.
- L_LEAVE
  Currently this message is ignored by the LLS. In future versions it could include some options and, of course, unpublish all blocks published by the client which leaves the network.
- L_PUBLISH blockID
  Add the reference to the local list of hosts which have the block. It replies with an empty L_PUBLISH_REPLY command which could include some options in future versions. If it is the first reference to the block on the local network, it will be published on the global network.
  The GLS's reply is a L_PUBLISH_REPLY command that is ignored in this version.
- L_UNPUBLISH blockID
  Remove the reference to the local list of hosts which have the block. It replies with an empty L_UNPUBLISH_REPLY command which could include some options in future versions. If it is the last reference to the block on the local network, it will be unpublished on the global network.
  The GLS's reply is a L_UNPUBLISH_REPLY command that is ignored in this version.

- L_LOOKUP blockID
  If the block is not available locally, it sends a L_LOOKUP command to its GLS. This message is replied by a L_LOOKUP_REPLY command with a list of clients which have the block. In future versions it could include some options such as minimum/maximum number of peers, only local lookup, etc. It also could reply with a list of LLS instead of a list of peers in order to make lighter LLS' task (see next point).

- L_LOOKUP_REPLY blockID list_of_hosts
  When the LLS sends a L_LOOKUP command to its GLS, it replies this command. The *list_of_hosts* is a list of LLS which have information about the block's location. Since the LLS has to reply with a list of clients, it chooses a LLS –the best network according to some measurement as latency, bandwidth, cost, etc.– and send a L_LOOKUP command to it. That LLS sends back a L_LOOKUP_REPLY command with a list of hosts that store the data. This list is sent to the client which requested the information in another L_LOOKUP_REPLY command. The list of host could be sorted before sending to the client; in that case, the best host cames first.

As it is written above, a LLS receives L_LOOKUP commands from other LLSs. The process of these commands is the same than when one of its client does. The difference is that LLS cannot forward the L_LOOKUP command to its GLS due to it would produce a loop.

B.1.3 Global Lookup Server

In a Chimera/Tapestry network, each blockID has a node responsible for it, that node –a GLS– is its root. When any node sends a message, it is routed to the root. If the receiver ID is a node, this node will receive the message; if the receiver ID is a blockID, the node that is responsible for the blockID will receive the message. In addition, each node in the path can manage the message to store useful information in its cache and it can also return cached results.

A GLS has two messenger interfaces: the local interface is connected to its LLS and the global interface is connected to a Chimera/Tapestry network.

The local messenger interface is very similar to a LLS, it receives these commands from its LLS:

- L_JOIN
  Currently this message is ignored. In future versions it could include some useful information.

- L_LEAVE
  Currently this message is ignored by the LLS. In future versions it could include some options.

- L_PUBLISH blockID
  Send a L_PUBLISH message to the Chimera network with its LLS' address. It replies with an empty L_PUBLISH_REPLY command which could include some options in future versions.

- L_UNPUBLISH blockID
  Send a L_UNPUBLISH message to the Chimera network with its LLS' address.

It replies with an empty L_UNPUBLISH_REPLY command which could include some options in future versions.

- L_LOOKUP blockID
  Send a L_LOOKUP message to the Chimera network. It replies with an L_LOOKUP_REPLY command with a list of LLSs which know the location of the requeested block. It could include some options in future versions.

The global messenger interface uses Chimera network where each message carries only one command. This message is routed to the closest node of a ID given. These are the messages received by a Chimera node:

- G_PUBLISH blockID reference
  Each node in the path adds the reference to its cache. There is no reply for this message.
- G_UNPUBLISH blockID reference
  Each node in the path deletes (when exists) the reference to its cache. There is no reply for this message.
- G_LOOKUP blockID sender
  Each node in the path put the references it has in the message. Once the message reach the root, it sends a G_LOOKUP_REPLY message to the sender. Note that the list of hosts is sorted, the closest references cames first. It could include some options in future options, the obvious one is *maximum number of references/hops*. It will make the lookup faster by taking cached list of references from the closest Chimera nodes.
- G_LOOKUP_REPLY blockID list_of_hosts This is the reply of a L_LOOKUP message. The list of hosts is put into a message which is sent to its LLS. Note that a L_LOOKUP message from a LLS might have several blocks as argument, however, a Chimera message has just one. The GLS fills the L_LOOKUP_REPLY in order; that means the faster the reply, the sooner it is put into the message. In the current implementation is not very relevant because it always has to reach the root; however, using some options a fast response might mean there are close sources of this block and the message will be auto-sorted.

B.1.4 Two Examples of Publication

The following examples show how Unpublish procedure is quite similar.

Data Locally Available

- Client sends to the Local Lookup Server a L_PUBLISH message.
- Local Lookup Server receives the message.
- Local Lookup Server checks its database and find some entries of the block on it.
- Local Lookup Server adds the new host to the list of host on its database.
- Local Lookup Server sends back a L_PUBLISH_REPLY message within the right code.
- Client receives the message.

Data Not Locally Available

- Client sends to the Local Lookup Server a L_PUBLISH message.
- Local Lookup Server receives the message.
- Local Lookup Server checks its database, but it does not find any reference to the block.
- Local Lookup Server adds the new host to the list of host on its database.
- Local Lookup Server sends a L_PUBLISH message to its Global Lookup Server.
- Global Lookup Server adds the new network to the list of networks on its database.
- Global Lookup Server sends a publish message to the Chimera/Tapestry network and send back a L_PUBLISH_REPLY message to the client[1].
- Each Global Lookup Server in the route to the root node[2] adds the reference in their database.
- Client receives the list of hosts that store the data.

B.1.5 Three Examples of Lookups

Data Locally Available

- Client sends to the local server a L_LOOKUP message.
- Local Lookup Server receives the message.
- Local Lookup Server gets the list of hosts which store the data from its database.
- Local Lookup Server sends back a L_LOOKUP_REPLY message within the list of hosts.
- Client receives the list of hosts that store the data.

Data Available on Another Network

- Client sends to the local server a L_LOOKUP message.
- Local Lookup Server receives the message.
- Local Lookup Server checks its database, but it does not find any host.
- Local Lookup Server sends a L_LOOKUP message to its Global Lookup Server.
- Global Lookup Server sends a lookup message to the Chimera/Tapestry network.
- Each Global Lookup Server in the path to the root node[3] checks their database.
- The first Global Lookup Server which has an entry of the block requested in its database sends back a G_LOOKUP_REPLY message within the address of some Local Lookup Servers.

[1]Depending the policy layer, the system could block contents from cheater nodes or when a court order forces an ISP to block some contents such as children pornography.

[2]See Section 8.6.1 to get more information about Tapestry and the root node.

[3]See Section 8.6.1 to get more information about Tapestry and the root node.

- Global Lookup Server receives the G_LOOKUP_REPLY message.
- Global Lookup Server applies its policy in order to put the best network first –in economical, latency or other measurement–, remove untrusted networks or whatever; and sends back a L_LOOKUP_REPLY message to its Local Lookup Server.
- Local Lookup Server receives the list of Local Lookup Servers in whose networks the data are.
- Local Lookup Server sends a L_LOOKUP message to one or more hosts of the list.
- The other Local Lookup Server gets the list of hosts which store the data and sends back to the Local Lookup Server in a L_LOOKUP_REPLY message.
- Local Lookup Server checks its policy to cache the content or not.
- If the content is cached, the list of hosts is sent to the cache and the address of the cache is sent to the client. Else, the list of hosts is sent to the client in a L_LOOKUP message.
- Client receives the list of hosts that store the data.

Data Not Available on the Global Network

It is similar to the previous example, however, the root node on the Tapestry network sends back an empty list that is sent to the Local Lookup Server and to the client later.

B.2 The storage system

B.2.1 Client

A client sends these commands:

- L_STORAGE_OFFER size
  The client offers size bytes of storage. This is sended to its LLS.
  There is not reply for this message.
- L_STORE blockID size
  The client which sends this message request data storage. This command might be sent to its LLS or to another client.
  It is not necessary that the client has the block. However, the block must be available on the network. This message is replied with a L_STORE_REPLY command. Currently this is ignored due to the information involved here will be processed by the policy layer.

A client receives these commands:

- L_STORE blockID size
  When this command cames from its LLS, this client accepts the proposal and reply with an empty L_STORE_REPLY. In future versions it could include some extra information, error codes, etc. It looks up and downloads the block in the usual way. In future versions a client could have it own policy and accept/refuse storage request from other clients.

B.2.2 Local Lookup Server

A Local Lookup Server receives commands from its clients:

- STORAGE_OFFER size
  The LLS takes these commands in order to manage the information relative to how much storage space is available on its local network.
  There is not reply for this message.
- L_STORE blockID size
  The LLS checks its policy and sends L_STORE commands to its clients –local storage– and/or to its GLS –global storage–. In future versions a L_STORE command will specify some parameters such as the number of replicas (locally and globally), how long they have to be available, etc. Currently this LLS replies the client's request with an empty L_STORE_REPLY command. It will include useful information such as how many replicas had been stored, where, when they will expire, etc.

It also receives L_STORE commands from its Global Lookup Server:

- L_STORE blockID size
  The LLS checks its policy and sends L_STORE commands to its clients –local storage– and/or to its GLS –global storage–. Currently, the policy is to forward the message to a client which offered space enough. In future versions a L_STORE command will specify some parameters such as the number of replicas (locally and globally), how long they have to be available, etc. Currently this LLS replies the client's request with an empty L_STORE_REPLY command. It will include useful information such as how many replicas had been stored, where, when they will expire, etc.

B.2.3 Global Lookup Server

A Global Lookup Server receives L_STORE commands from its LLS:

- L_STORE blockID size
  Send a G_STORE message to the Chimera network. Reply with a L_STORE_REPLY command.

It also receives messages from Chimera network:

- G_STORE blockID size
  Send a L_STORE command to its LLS. Reply with a L_STORE_REPLY command.

B.2.4 Example of Storage Requests

Depending the configuration of the policy, the Local Lookup Sever could manage STORAGE_OFFER messages' information of its clients and distribute the data between them. In exchange, these clients could get some advantages from the ISP such as privileged access to the cache, the availability of store contents on it or other extra services.

Client to Client

- A client sends a L_STORE message to another client.
- The later receives the message and checks the message's policy options and its policy configuration.
- Depending the policy, it sends back a L_STORE_REPLY message and performs some operations (e.g. lookup+download) when this node accepts the storage demand or nothing at all otherwise.

Distributed and Local Storage

Depending the message's policy options –local/global, number of replicas, storage period, etc.– a store message is managed in one or other way. This system is flexible enough to support a wide range of options. The desirable behaviour will be implemented in the policy layer and it will be independent from the data layer.

This is the trace of a store procedure:

- The client sends a L_STORE message to its Local Lookup Server (the message includes the policy options)
- The Local Lookup Server checks its policy configuration and sends –if necessary– a l_store message to its Global Lookup Server.
- The Global Lookup Server checks its policy configuration and sends –if necessary– one or more[4] store messages to the Chimera/Tapestry network.
- Depending the policy, some Global Lookup Servers can accept the storage request and send back a store_reply message to the sender.
- The Global Lookup Server gets STORE_REPLY messages and sends a L_STORE_REPLY message to its Local Lookup Server within all the responses together.
- The Local Lookup Server sends back a L_STORE_REPLY to the client.

B.3 Data Transfer

Clients transfer data to each other. After finding the client which actually has a block, the download can start. There are transfers between clients, however, Local Lookup Servers can send and receive data in order to cache them.

A simple own-made transfer protocol is used in data transfers. It is because it was easy to implement and it gives us an homogenous message format without the need of an external server. It could be changed in the future versions if more features are needed.

A client sends/receives messages to/from other clients. The commands are:

- L_DOWNLOAD blockID
  The client asks another client for a block.
  In future versions it could include some options such as offset, length of the piece of the block, etc. The reply is an UPLOAD command.

[4]It depends on how many replicas are demanded.

- L_UPLOAD blockID length binary_data
  This client sends the data of the *block* requested (*length* bytes). When the block requested is not available, *length* will be 0. In future versions it could include some options such as offset, total length of this block, etc. There is not reply for this command.

B.3.1 Trace of a data transference

- Client A sends a L_DOWNLOAD message to client B. It specifies the blockID it wants to download.
- B checks the availability of the data and sends back a L_UPLOAD message within the length of the data transfered –zero when the block is not available– and the binary data of the block.

Appendix C

GNU Free Documentation License

Version 1.2, November 2002
Copyright ©2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

2. VERBATIM COPYING

3. COPYING IN QUANTITY

4. MODIFICATIONS

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

ADDENDUM: How to use this License for your documents

Bibliography

[1] Samer Al-Kassimi. Evaluation of a scalable peer-to-peer lookup protocol for internet applications. 2005.

[2] CacheLogic. Peer-to-peer in 2005, 2005. <http://www.cachelogic.com/research/p2p2005.php> (20 February 2006).

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[4] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays, 2003.

[5] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[6] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller. Hierarchical p2p systems. In *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, Klagenfurt, Austria, 2003.

[7] K. HILDRUM, J. KUBIATOWICZ, S. RAO, and B. ZHAO. Distributed object location in a dynamic network, 2002.

[8] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[9] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[10] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing.

[11] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for dhts: Some open questions. Technical report, 2002.

[12] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[13] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment, 2003.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.