Distributed Computing in Peer-to-peer Networks

Emir Ahmetspahić

LITH-IDA-EX–04/095–SE

2004-09-27

Linköpings Universitet
Institutionen för datavetenskap

Final Thesis

Distributed Computing in Peer-to-peer Networks

by

Emir Ahmetspahić

LITH-IDA-EX–04/095–SE
2004-09-27

Supervisor: Niclas Andersson
Examiner: Nahid Shahmehri

LINKÖPINGS UNIVERSITET	Avdelning, Institution Division, Department Institutionen för datavetenskap 581 83 LINKÖPING	Datum Date 2004-09-27

Språk Language Svenska/Swedish X Engelska/English	Rapporttyp Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport ——	ISBN ISRN LITH-IDA-EX--04/095--SE Serietitel och serienummer ISSN Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/ida/2004/dd-d/095/

Titel Title	Distributed Computing in Peer-to-peer Networks
Författare Author	Emir Ahmetspahic

Sammanfattning
Abstract
Concepts like peer-to-peer networks and distributed computing are not new. They have been available in different forms for a long time. This thesis examines the possibility of merging these concepts. The assumption is that non-centralized peer-to-peer networks can be used for effective sharing of computing resources. While most peer-to-peer systems today concentrate on sharing of data in various forms, this thesis concentrates on sharing of clock cycles instead of files.

Nyckelord
Keyword
Peer-to-peer, Distributed Computing, Peer-to-peer networks, Job distribution

Abstract

Concepts like peer-to-peer networks and distributed computing are not new. They have been available in different forms for a long time. This thesis examines the possibility of merging these concepts. The assumption is that non-centralized peer-to-peer networks can be used for effective sharing of computing resources. While most peer-to-peer systems today concentrate on sharing of data in various forms, this thesis concentrates on sharing of clock cycles instead of files.

Contents

Preface

This chapter contains background information about this master thesis. Also there is contact information for the author and supervisors of this thesis.

About report

During the fall of 2002 I had several ideas about what to do for my master thesis. All these ideas had one thing in common - distributed computing. In November 2002, I contacted Niclas Andersson at National Supercomputing Centre (NSC) in Linköping. After meeting with Niclas Andersson and Leif Nixon my ideas were somewhat modified. In late October 2003 I started working on my master thesis. The result of this work is presented in this report.

About the author

The author of this report is Emir Ahmetspahić. I am currently completing my final year as a student in the Master of Science Programme in Communication and Transportation Engineering at Linköping University. I can be reached at the following e-mail address: emiah182@student.liu.se

About the supervisors

The supervisor of this master thesis was Niclas Andersson. He currently works as a parallel computing expert at the National Supercomputing Centre (NSC) in Linköping, Sweden. Niclas can be reached at the following e-mail address: nican@nsc.liu.se

Formal examinator of this thesis was Nahid Shahmehri. She is professor in computer science at the Department of Computer and Information Science (IDA), Linköping University in Linköping, Sweden. She is currently also director of the Laboratory for Intelligent Information (IISLAB). She can be reached at the following e-mail address: nahsh@ida.liu.se

Acknowledgements

First and foremost I would like to thank Niclas Andersson. He has been extremely helpful and always took the time to point out errors and explain different concepts I did not understand previously. Also I would like to thank Leif Nixon who corrected countless mistakes in English grammar that were present in the first draft of this report. He also offered many useful programming advices. Finally, I wish to thank all other personnel at the National Computing Centre in Linköping that made my stay there a pleasant one.

Chapter 1

Introduction

As it can be noticed by the title of this report, this study concerns itself with the distribution of computing power in peer-to-peer networks. This chapter presents the goal of the thesis and requirements that need to be satisfied so that this master thesis can be considered successful. Also there is a short discussion of requirements and limitations.

1.1 Goals and Requirements

The goal of this thesis is to present a system, which is going to enable effective use of distributed computing. This system needs to satisfy the following requirements:

1. The system should lack any centralized services.
2. The system should provide full security model.
3. The system should have full brokering capabilities.

The thesis is divided into three parts. First there is a survey of already existing systems. Having a survey of existing systems has two important benefits. Firstly, because of time constraints involved with this thesis it is important not to waste any time on solving problems that have already been successfully solved by others. Secondly, by studying other systems, errors and mistakes made by the authors of these systems are brought to the attention and thus are not going to be repeated.

In the second part of the thesis, a system that satisfies all the requirements is going to be created. The system is going to be designed in three layers where each layer satisfies one of the three of the above mentioned requirements: Network, brokering and security. Some of the ideas incorporated into these three layers are going to be borrowed from other successful systems for distributed computing and peer-to-peer networking. There is no need to create new solutions for the problems, where satisfying solutions already exist. Other times when there are no satisfactory solutions available, new solutions are going to be designed. This system is then going to be scrutinized with the great care so that it can be made sure that there are no obvious flaws. In the third part of the thesis, this system is going to be implemented.

This report mimics these three phases that this thesis experienced. The first part concerns itself with examination of related works and explanation of concepts that the reader needs to understand to be able to fully enjoy this report. The second part of the report presents the system through three subchapters. Each subchapter presents a layer of the system. And lastly there is a presentation of the code.

1.2 Lack of Centralized Services

The system is fully distributed i.e. there are no central services. It is built upon a network architecture known as peer-to-peer. This enables the system to treat all the participating nodes equally. There are no clients or servers. Everyone provides and uses resources after their own possibilities and needs. Scaling problems which usually appear in client-server architectures are also avoided.

1.3 Full Security Model

Every network is the sum of its users. If users think that joining a network is going to pose a risk to their data or computers they are not going to join the network. Thus security threats need to be handled by the system. Users should be protected as much as possible from their malicious peers. This report contains formal examination of threats to the system and presents possible solutions to these threats.

1.4 Full Brokering Capabilities

For a user to join a network there needs to be some incentive. In a system built for distributed computing, the incentive is that the user, after joining a network can use idle computers owned by other peers on a network. When the user is not using his computer he or she provides his or hers machines to other peers on the network. To be able to efficiently share resources between peers on the network there needs to be some means to easily describe job and machine requirements. These requirements are later used for matching jobs and machines. One of the integral parts of this thesis is to provide such mechanisms that efficient sharing of resources among peers is made possible.

1.5 Limitations

Because of the sheer amount of work needed for creation of the system for distributed computing, there are some compromises that are going to be made. Compromises were made mainly in two areas because of the lack of time. These two areas are security and brokering. A full security model is going to be created for the system but it is not going to be implemented in code. In the area of brokering a decision has been made that system is not going to support queuing. Jobs can not be queued for later execution. Nodes are either available (and accept jobs) or they are busy (and do not accept jobs).

Chapter 2

Concepts and Related Works

There are dozens and dozens of different computing concepts. The Reader of this report needs to understand some of these to be able to fully understand the master thesis presented in this report. This chapter presents these concepts. Some readers are probably already familiar with these concepts but they are presented here anyhow because of their importance. Also, there is presentation of previous work in the fields of peer-to-peer computing and distributed computing.

2.1 Distributed Supercomputing

Traditionally computing centres around the world have used big computing machines, so called supercomputers to fulfil their computing needs. Usually these computing machines were built in comparably low quantities. Machines from different companies were incompatible with each other's. Different companies used different architectures and different operating systems. Users were usually locked into solutions from one company without the possibility of smooth change. All these different factors contributed to the high price of the traditional supercomputers. In the mid-nineties the situation began to change.

During the late eighties and early nineties computers moved into ordinary people's homes. There were several reasons for this but perhaps the main reason was the success of the Internet among the general public. The home market was big in units but was not as profitable as selling supercomputers to big companies and institutions. Companies that wanted to stay afloat usually had to sell many units. That was the one of the reasons that many big companies at the beginning did not pay much attention to this market. In the eighties home market was fragmented but as the nineties came Intel based personal computers (PC) became dominant. As the home market exploded in the nineties Intel profited greatly. Soon no other computer company could invest as much as Intel in R&D. Intel's x86 architecture, which was technically inferior to some other architectures, was soon surpassing all other architectures when it came to price to performance ratio. Computers sitting at people's desktops were faster than supercomputers ten years before. The realization that these computers could be stacked together into clusters and the availability of comparably easy to use software for clustering, like Beowulf, started eroding the positions of monolithic supercomputers at computing centres. At the same time there was a realization that computers sitting on people's desktops at work, or at home were idle most of the time. Many companies and individuals realized that this was a huge untapped resource.

At various universities and companies around the world there was a febrile activity among researchers. Different ideas were tested. Some people were working towards uniform access to different clusters and supercomputers while others worked on harvesting the power of idle desktops. Soon different systems started popping up. Some like ever popular SETI@home and distributed.net exploited idle desktops in a master-slave architecture. Others like Condor used a centralized broker while allowing all the nodes to act as workers and masters. Perhaps most revolutionary of all these new ideas was the concept called Grid.

The computing Grid is a concept modelled upon the electric grid. In the electric grid, electricity is produced in power plants and made available to consumers in a consistent and standardised way. In the computing grid, computing power is to be produced instead of electricity and is also to be made available to consumers in a consistent and standardised way. According to Ian Foster [7] a grid is a system that:

1. Co-ordinates resources that are not subject to centralized control
2. Is using standard, open, general-purpose protocols and interfaces
3. Delivers non-trivial qualities of service

Currently there are several large-scale grid deployments. Some of these are GriPhyN, EUGRID, NorduGrid and TeraGrid. The Globus toolkit, which is used in some form by all the four previously mentioned projects, has become the de facto standard of the Grid world.

2.1.1 Related Works

There are several dozens of different systems for distributed computing. Most of these systems are only created for specific tasks and lack brokering environment. One such system is distributed.net. Other systems have full brokering capabilities. Two of these systems for distributed computing are presented in next two subchapters.

Condor

Condor started its life as a part of the Condor Research Project at the University of Wisconsin-Madison. The goal of the project is to develop mechanism and policies that would enable High Throughput Computing on large collections of distributively owned heterogeneous computing resources.

Condor organizes machines in pools. Such condor pool consist of resources and agents. Agents are machines used by the users to submit jobs while resources represent machines that the jobs can be executed on. Depending on its setup a single machine can act both as an agent and as a resource. Every Condor pool contains a matchmaker. The matchmaker is a single machine that acts as a central manager of resources inside the pool. Resources and agents send information about themselves to a matchmaker.

Information sent to the matchmaker is in the form of the classified advertisments (ClassAds). ClassAds are semi structured data structures that contain keys and corresponding values. The matchmaker's job is to match job ClassAds and resource ClassAds. If a match has been made between an agent and a resource, the matchmaker will contact both of them and notify them of a match. The agent now contacts the resource to verify that the match is still valid. If the match is validated, agent involves claiming protocols.

Jobs can be executed in six different environments. These environments are called universes. Different universes fit different types of jobs. Prior to submitting a job a Condor user needs to define his job. This is done with the help of a submit file. The submit file is semi structured data file otherwise known as dictionary. It contains key-value pairs. In the Condor literature keys are usually referred to as attributes. A simple job submit file looks like this:

```
MPI job description (4 nodes)

universe      = mpi

executable    = some_mpi_job

requirements  = Memory > 256

machine_count = 4

queue
```

The universe attribute in above example declares that the job is to be run in the MPI universe. The machine_count attribute specifies how many nodes are to be used for the execution of this job. (More thorough information on Condor's execution environments and matchmaking can be found in the articles Matchmaking: Distributed Resource Management for High Throughput Computing [1] and Distributed Policy Management and Comprehension with Classified Advertisments [15]).

Distributed Network

Distributed.net is a volunteer project for distributed computing created in 1997. The Distributed.net system, unlike the previously mentioned Condor system, lacks a general execution environment. All jobs need to be specifically tailored for the network. When a new project becomes available users need to download a new version of the software. Such a new version contains a module that allows for processing of the new project's work units.

Distributed.net is built upon a client-server architecture. At the top of the system there is a master server which is also known as a master key server. It issues and keeps track of issued work units. Under the master server there are proxy key servers which request large blocks of work units from the master server. These large chunks, known as super blocks are split into smaller blocks - work units, which are sent to requesting clients. The Distributed.net network architecture is illustrated in figure 2.1.1

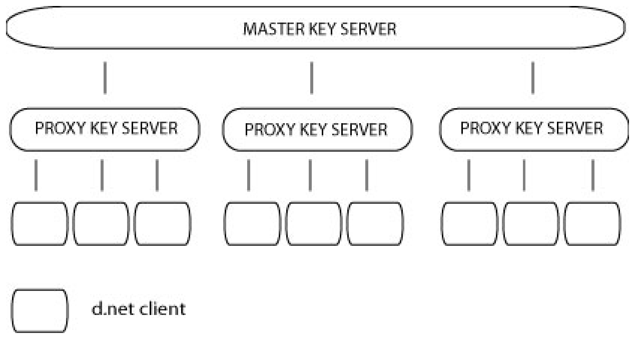


Figure 2.1: Overview of distributed.net architecture

All jobs are centrally managed and can not be created by the clients. The clients only purpose is to serve the master key server with raw processing power. After clients' finish processing work units they send them back to proxy key servers which in turn send them back to the master key server.

2.2 Peer-to-peer Networks

It is hard to a find single point in history, which could be characterized as the starting point of the service architecture type known as peer-to-peer. Different people would probably point out different points in history. What some may consider as a very important paper on peer-to-peer, others will tend to diminish as unimportant. What is certain is that the history of peer-to-peer networking is closely intertwined with the history of the network that is today known as the Internet.

As hard as it is to find a single starting point for peer-to-peer, it is even harder to find a starting point for the Internet. Some people find that J.C.R. Licklider's publication of paper [3] on how interaction between humans can be enhanced by creating computer networks, is a stepping stone of Internet. Licklider was the first chief of Defence Advance Research Project Agency, also known as DARPA. The predecessor of today's Internet, ARPANET was launched by the agency in 1969. ARPANET had several services that had peer-to-peer like functions and thus can be considered as a stomping ground of peer-to-peer architecture.

2.2.1 Service Architectures

Network services can roughly employ two different architectures for communication between computers on the network. The first architecture type is known as client-server architecture and is widely employed on the Internet today [6]. The main characteristic of client-server services is its distinction between service providers, known as servers, and service requesters, known as clients. Usually there are a few big servers and many small clients. This can be seen in figure 2.2. At the top is one, big server that handles requests from many clients.

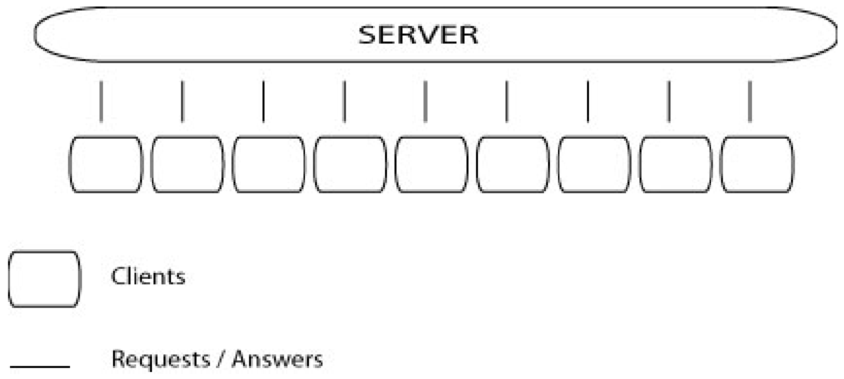


Figure 2.2: Client-server network

The easiest way to understand client-server architecture is to look closely at the media situation in today's society. There are few content providers (TV stations, newspapers, etc) and there are many content consumers (regular people). This is exactly the structure that the client-server architecture in different computer services mimics. Few servers (content providers) and many clients (content consumers). Depending on what type of service you want to create you might find this mimicking good or bad. Generally services that intend to provide information from centrally managed repositories are well suited for client-server architectures.

One such example is the Network File System or NFS [14]. NFS was introduced by Sun Microsystems in 1982 and has as its primary goal to allow effective sharing of files from a central location. This gives several advantages to NFS users. Among others, it decreases storage demands and allows for centrally managed data files. Figure 2.3 contains a simplified interaction between NFS client and file server.

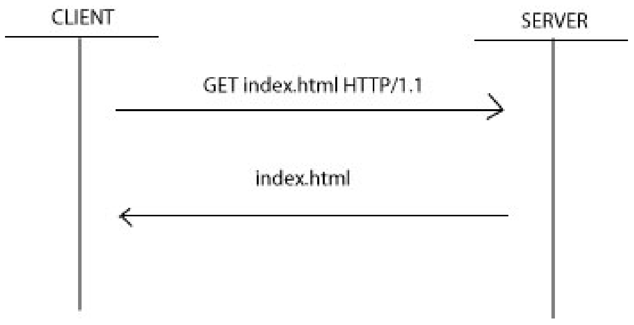


Figure 2.3: Communication between NFS server and client

In contrast to client-server services, peer-to-peer services do not create an artificial difference between servers and clients. All computers on peer-to-peer network are equals and can both provide and request information. A real world equivalence of this architecture is ordinary people meeting each other in a town square. They can either choose to talk to (provide content) or to listen to (download content) other people. This is illustrated in figure 2.4.

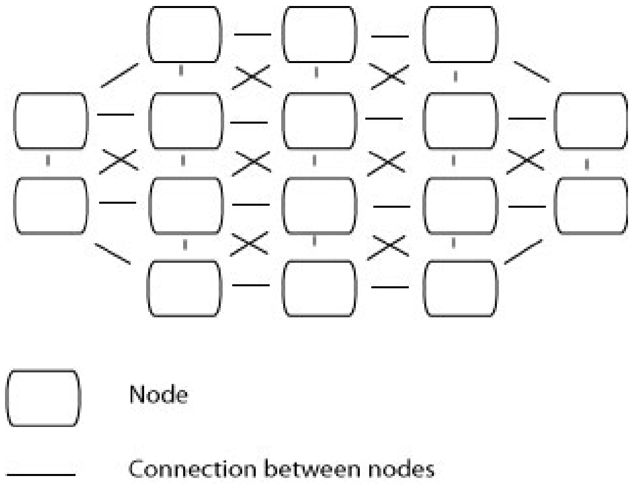


Figure 2.4: Peer-to-Peer network

It is important to notice that the border between these architecture types is not sharp. Different experts are going to give different answers when asked what exactly peer-to-peer is. With the arrival of new services this border is going to get even more blurred.

2.2.2 Internet History Through Different Architectures

Originally the Internet predecessor, ARPANET connected four different computing sites. These four sites were already established computing centres and as such were to be connected as equals. They were peers.

Services used on ARPANET were usually built with client-server architecture. Although this architecture was used, hosts did not act only as clients or servers, which is the case with most of the hosts on today's Internet. Hosts on ARPANET were usually both servers and clients. Computers would usually act one time as clients, just to assume the role of server next time someone was accessing the computer. Usage pattern between hosts as whole was symmetric [6]. Because of this early Internet is usually considered as peer-to-peer network.

ARPANET quickly became popular and started growing immensely. Explosive growth radically changed the shape of the network. What started as a network between few computing sites became a natural part of every home in the developed world. This growth was not only positive. Several problems appeared. Most of these problems caused the Internet to become less open. Less open networks favour client-server type services over peer-to-peer services. Below is a list of some of these problems.

Unsolicited advertisements, otherwise known as spam were literally unknown on Internet before the Internet boom in the early nineties. Today they have overrun early peer-to-peer like services as Usenet. Lack of accountability in early peer-to-peer services as Usenet and e-mail make them a popular target for unscrupulous people that send out spam.

Another change that appeared in the early nineties was decreased reachability between hosts. Originally every host on the Internet was able to reach every other host. A host that can reach the Internet was also reachable by computers on the Internet. This suited well symmetric usage patterns that dominated on early Internet. This was about to change in the early nineties. Firstly, system administrators started deploying firewalls, which are basically gates between different parts of networks. With the help from firewalls the system administrator can control the traffic flow to, and from the Internet. Usually they allow users on internal networks to reach the Internet while they disable access from the Internet to computers on local networks. This is done to increase security level on internal networks and is as such a very useful tool. Another change that affected reach-ability on the network was the deployment of dynamic IP addresses. Because of shortage of IP addresses dynamic IP assignment became norm for many hosts on Internet. An individual computer receives different IP addresses by its Internet service provider (ISP) every time it connects to the Internet.

As mentioned previously all of these changes favoured client-server architecture type for services over peer-to-peer like services. This caused Internet usage patterns to change from symmetric to asymmetric. Many users downloading data and few, big service providers.

Then in the late nineties interest in peer-to-peer like services surged again. It was mostly fuelled by the success of Napster. In the wake of the Napster trial and shutdown of the original Napster network, several more or less successful peer-to-peer services followed.

2.2.3 Related Works

The next four subchapters are going to examine four different peer-to-peer like services. The first two, Usenet and DNS were services that already appeared on early Internet while the other two, Napster and Gnutella are latecomers to peer-to-peer world.

Domain Name System

The Domain Name System or DNS [11] [13] for short was introduced in 1983 as an answer to the Internet's growing problems. Originally every host on the Internet contained a flat text file known as hosts.txt. This file contained the mapping between IP addresses like 130.236.100.21 and more user friendly names like nsc.liu.se. This file was copied around the Internet on regular basis. Managing this file became more and more tedious as Internet grew from few hosts to thousands of hosts. At the end, maintaining a accurate host list became almost impossible. DNS was the answer to this problem.

With the introduction of DNS, several peer-to-peer like structures appeared. Firstly, DNS introduced a hierchical system for storing records about hosts on the Internet. There is no single host that contains information about all the other hosts. Thus there is not a single failure point as is the case in client-server systems. The name servers at Linköping University are responsible for host names in liu.se zone. They do not have any authorative information about any other zones. If there is a hypothetical power outage at Linköping University that affects the name servers, only host names under liu.se are not going to be resolvable. This power outage is not going to disable name resolving of hosts outside liu.se zone. Figure 2.5 contains simple illustration of different zones on Internet.

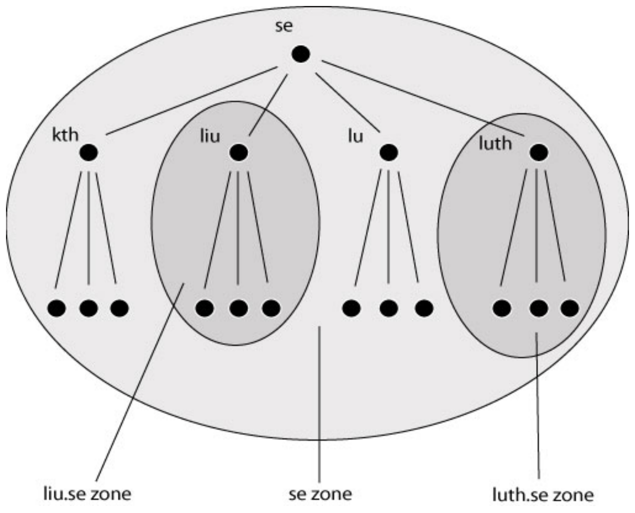


Figure 2.5: DNS Zones

When a user types a host name in his web browser, his computer is going to try to translate the host name into an IP address. His computer contacts its local name server and supplies it with a host name it needs to resolve. It is assumed that the host name that is being searched is example.lu.se. Now the local name server contacts a root name server and asks him about the IP number of example.lu.se. As an answer it receives a referral to a name server that is responsible for the se zone. The root name server does not know the IP number of example.lu.se, but it knows that the se name server knows more. The se name server can not resolve searched host name either so it sends a reference to the name server that is responsible for the lu.se zone. Finally the name server at lu.se can answer the query. The local name server receives the answer and sends it back to the user's computer. This process is illustrated in figure 2.6.

Another peer-to-peer like structure that DNS introduced is caching of data. The above mentioned local name server is not going to throw away the mapping between example.lu.se and IP address after sending the answer to the requesting user. It is going to keep the data for some time. This is done in case that some other user requests this information again. Next time someone requests this data, the local name server is going to be able to

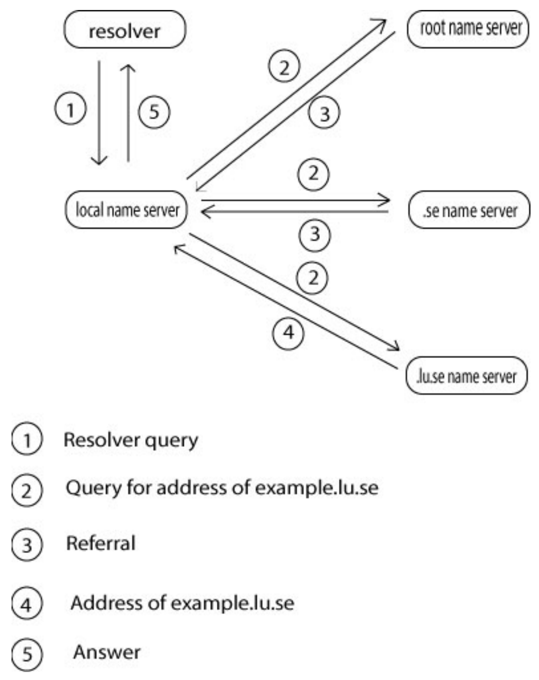


Figure 2.6: DNS Query

answer directly without having to query other name servers. Data is moved close to the requesting computer. This reduces load on the name servers and the amount of data traffic flowing in and out from name servers.

Usenet

Usenet [2] was for a long time one of the Internet's most popular services. It is still in use today although its popularity has decreased. This is one of the services that has been greatly hurt by the tremendous growth of spam. Users on the Usenet network can post their messages in different groups, which are also known as news channels. These messages are then spread among different Usenet hosts who choose which news channels they want to subscribe to.

Introduced in 1979, Usenet originally used Unix-to-Unix copy protocol (UUCP) [9]. Computers running UUCP would automatically contact and exchange messages with each other. Today Usenet has switched to a TCP/IP based protocol known as the Network News Transfer Protocol (NNTP) [10].

NNTP implements several peer-to-peer like functions. For example every message stores in its header, the path it has taken. If news server A sees that a message has already passed through news server B, it is not going to try to send the message to news server B.

Napster

Napster is a proprietary file sharing service that allowed users to swap their music files with each others. It is probably the main reason that peer-to-peer architecture experienced a revival in the late nineties. This document describes the original Napster software that allowed users to swap their music files with each other. After various legal troubles this version 1 of Napster was closed in February 2001. The software was rewritten and released as Napster v2.0 which is essentially an online music store. As Napster was proprietary software and not freely available this chapter had to be based on reports [5], [4] written by people that had reverse engineered the protocol. These reports are not official and may or may not contain errors.

Napster uses a dual architecture network for distribution of music files. It is an example of the previously mentioned systems that blurs the line between peer-to-peer and client-network architectures. File indexing in Napster network is done via a simple-client model while file exchange is handled directly by the peers on the network. This model, which is pictured in figure 2.7, is also referred to as a centralized peer-to-peer service.

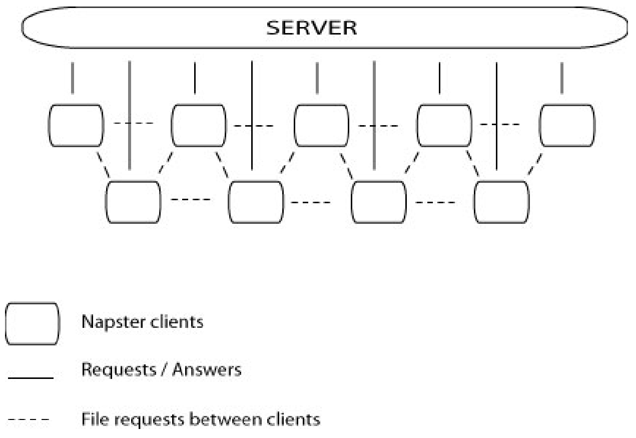


Figure 2.7: Napster architecture

To gain access to Napster network the user needs to register first. Registering is done automatically from the Napster client. User names on the Napster network are unique and are used in conjunction with a password to uniquely identify every user. This identification is done by Napster server. After successful logon onto the network, clients automatically upload their list of shared files to the main Napster server. The server indexes this file list into its database. When a specific client needs to find some file it issues a query to the Napster server. The server parses its database. In case the file is found, the file location is returned. The client is now free to request the searched file from the peer that has it. The download procedure varies a bit depending on if the peer with the requested file is firewalled or not.

Gnutella

Gnutella is an example of a decentralized peer-to-peer service. Unlike Napster there is no central repository of information on the Gnutella network. Searching and downloading is completed by peers on the network querying its neighbours. This description is based on the draft of Gnutella Protocol Specification 0.6 [12].

The Gnutella network is usually referred to as GNet [12]. Nodes participating in GNet are referred to as servents, which is a contraction of SERver and cliENT. Because the Gnutella network lacks any fixed structure, servents need to get the host address of at least one other servent to be able to connect to the network. Finding out servent addresses can be done in two different ways. The servent can save peer addresses while present on the network and then use them next time it connects or it can use the GWebCache protocol which allows the servent to retrieve host addresses by sending simple HTTP queries to special GWebCache servers. After the address is obtained the servent connects to the network. Figure 2.8 contains a simplified handshake between a servent connecting to the network (client) and a servent already on the network (server).

Because of the lack of a central repository for information, Gnutella is almost impossible to shut down. The way the Napster service was shut down after Napster Inc. lost their case against RIAA (Recording Industry Association of America), would be impossible with Gnutella. This is Gnutella's greatest strength, and at the same times its greatest weakness. Although the lack of a central directory server makes Gnutella harder to shut down,

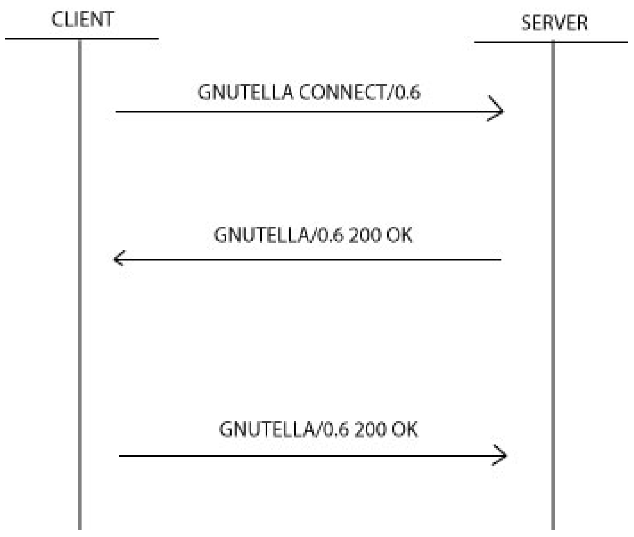


Figure 2.8: Gnutella handshake

it harms its performance greatly. Searching on decentralized peer-to-peer systems is simply never going to be on par with searching on centralized peer-to-peer systems.

If a servent on the network needs to find a specific file it issues a query message. This query message is sent to all of the servent's neighbours. Neighbours in turn forward the message to their own neighbours. This way the message is transported across the network. Usually after the message has been forwarded seven times it is discarded. Every node that receives a query message checks for the requested file and answers in case that they have the file. The requesting servent can now start downloading the file.

Chapter 3

Theory

This chapter is going to present a full description of the network that satisfies the requirements set in chapter 1. A system for distributed computing can be divided into four parts. These four parts or layers are:

- Communication
- Brokering
- Security
- Execution

The Communication layer concerns itself with how peers connect to the network and how they communicate with each other while being on the network. It also provides a mechanism to protect the network from fragmentation and provides a foundation for distribution of jobs across the network. This layer is the one that fulfils the first of the three requirements of this thesis - no central points in the network. The next subchapter contains a full explanation of the communication layer.

The brokering layer handles matching of jobs and machines. It includes the matching algorithm and an XML based language for description of jobs and machines. This layer fulfils the second requirement of this thesis - full brokering capabilities. Like with the communication layer there is a subchapter that describes this layer.

The security layer handles the security of the system. Together with the execution layer it fulfils the third and final requirement of this thesis - full security for the users of the system. The subchapter that describes this layer contains a formal examination of the threats and presents possible solutions to these.

The final layer - the execution layer handles execution of the jobs. It is partly described by a subchapter on security and partly described in the chapter that describes implementation of the network.

3.1 Communication

One of the requirements set for this system was lack of centralized points. Thus the main characteristic of the network that the system is run on has to be a lack of any centralized points. Such networks are usually called peer-to-peer networks. This network needs to be created for the purpose of distributing jobs.

The easiest explanation of how such a network is created is through examples. Thus there is going to be an extensive use of examples and figures in this subchapter. Figure 3.1 contains an illustration of a sample peer-to-peer network with ten nodes. This sample network is going to be used as the base of many hypothetical situations that are going to be examined in this subchapter. Nodes one to ten are connected to each other. Node N wants to join the network. To do this, node N has to know the address of at least one node already present on the network.

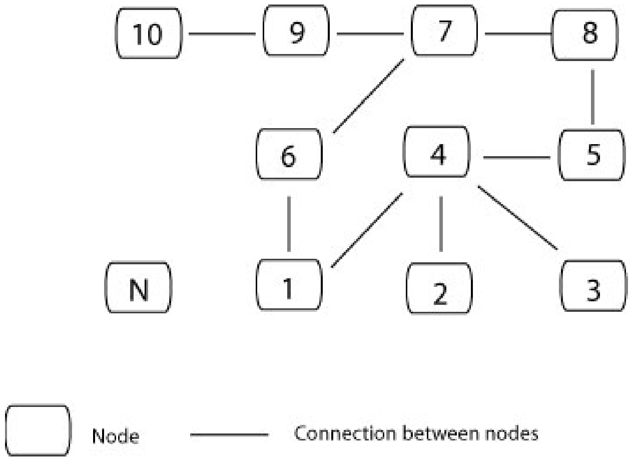


Figure 3.1: Example peer-to-peer network

Each node, already present on the network, is connected to one or more nodes. The owners of the nodes are the ones that decide how many nodes their own computer should be connected to. The general rule is that more connections to other nodes means more traffic. Nodes with low bandwidth to the rest of the world should minimize the amount of connections and hence minimize traffic flowing into the node. On the other end, by minimizing the number of connections, the owner increases the risk of his or her node being cut off from the network. For node 2 in figure 3.1 only node 4 has to disappear for it to be cut off from the network. If node 4 is to be cut off, four different nodes have to disappear. A greater amount of connections means also that a node can reach a greater number of hosts in fewer hops. In figure 3.1, node 4 can reach six nodes in two hops, while node 2 can only reach 4 nodes in the same amount of hops. Thus nodes with high bandwidth should preferably have many connections. Making high-speed nodes prefer having many connections has one pleasant side effect - creation of a backbone of high-speed nodes. This is illustrated in figure 3.2. High bandwidth nodes are located in the middle of the network while low bandwidth nodes are on the fringes of the network.

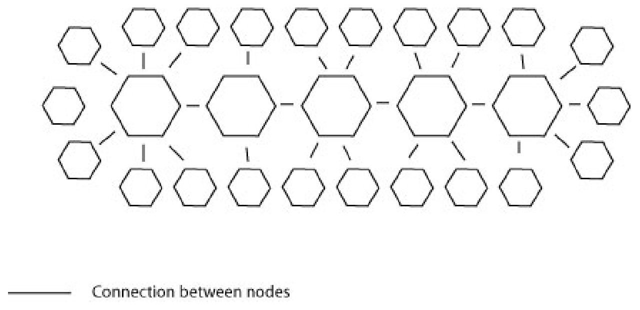


Figure 3.2: Network with high and lowbandwidth nodes

This structure is automatically created because high-speed nodes are going to strive to have many connections. This gives them benefits - high connectivity and decreased risk of being cut off from the network. At the same time low bandwidth nodes strive to be on the fringes of the network to avoid having too much traffic.

3.1.1 Finding Hosts

Like with Gnutella, there is no fixed structure on the network. Thus to connect to the network, the new node N in figure 3.1 has to know the address of at least one node already present on the network. Every node, aspiring to connect to or already connected to the network, stores a list of addresses of all known hosts. Nodes aspiring to connect to the network are going to check this list for entries. They are randomly going to choose one entry and try to connect to it. In the case of a failure, the connecting node is going to fetch a new entry and try instead to connect to that node. In the example from figure 3.1, it is assumed that node N's list of known hosts only contains one entry - the address of node 1. Thus to connect to the network node N will have to connect to node 1. After connecting to the network, nodes can expand their known hosts list either by asking other hosts or by listening to the packets traversing the network.

3.1.2 Establishing Connection

All nodes on the network have three tables that are of vital importance to the connection procedure. First of these three tables is the one already mentioned above - the list of all known hosts. As it is said, it is a list that contains the addresses of all hosts known to the owner of the list. The second of these tables is the list of permanently connected nodes. For example node 4 in figure 3.1 has nodes 1, 2, 3 and 5 in this list. Packets that are travelling on the network are forwarded and received from these nodes. And finally, the third table is the list of all temporarily connected nodes. The difference between permanently connected nodes and temporarily connected nodes is that later ones are only connected for some pre-configured time frame, while the first ones are connected indefinitely.

Nodes on the network do not accept nor forward packets to nodes that they are not permanently or temporary connected to. Packets that are directly received from other nodes are discarded.

The amount of nodes one host is connected to is regulated by the owner of that host. Based on the characteristics of the host, the owner chooses if the host should connect to few or many nodes. Besides setting the maximum amount of nodes its host should connect to, owner sets also the minimum amount of nodes his or hers host should connect to. When a host exceeds the maximum size of connections, it is not going to allow any more connections.

If a node on a network is connected to less nodes than the maximum size of connections allows, and at the same time is connected to more nodes than the minimum amount of connections, that node is not going to actively search after new nodes to connect to. Such a node is passively accepting connections. When the amount of connections falls under the value for the minimum amount of connections, the node starts actively searching for other nodes to connect to. This is the reason that the value for the minimum amount of connections is sometimes also referred to as the threshold between actively searching for and passively accepting connections. If the sample network in figure 3.1 is used again and there is assumption made that node 4 accepts at most five connections while the threshold value is three, node 4 is not going to connect to more than five nodes. At present it is also not going to try to connect to more nodes. Node 4 is already connected to four other nodes and that value is above the threshold value between passively accepting and actively searching. If node N tries to connect to node 4, node 4 is going to accept the connection. If node 2 and 3 somehow disappeared from the network, node 4 would switch to active mode and start trying to establish more connections.

3.1.3 Permanent Connections

To be able to explain the difference between permanent and temporary connections, there is again going to be an extensive use of the sample network in figure 3.1. It is assumed that node 1, which is connected to two other nodes, is allowed to connect to maximum three other nodes. Thus it is accepting new connections but is not actively searching for them. Threshold value is two.

Node N is, as can be seen in the figure, not connected to any nodes on the network. It is outside of the network. Thus its permanent connection list is empty. Its list of known hosts contains only one entry - node 1. To connect to the network node N will have to contact node 1 and try to establish connection with it. Upon contacting node 1, node N for the reasons explained later provides node 1 with information on how many nodes it is connected to and how many nodes it is allowed to connect to at most.

Node 1 looks into its permanent connection list, sees that it is allowed to connect to one more node and allows node N to establish connection with it. Such a connection is called a permanent connection.

3.1.4 Temporary Connections

But what would happen if node 1 was only allowed to connect to two hosts at maximum? Would node 1 just ignore node N's connection query? In this case N is never going to be able to connect to the network - it only knows of node 1.

Answer to the second question above is no, node 1 is not going to ignore node N's query. As mentioned in the previous chapter, node N is in its query sending information on how many other nodes it is connected to. Node 1 notices that N is not connected to anyone else - thus N is not part of the network. Because of this, node 1, which does not have place for any more permanent connections, allows node N to connect for some short time frame. This short amount of time can be used by N to find addresses of other nodes. N can then try to connect to these nodes and become a permanent part of the network. At the time the temporary connection dies, node N is hopefully going to have a permanent connection to some other node on the network. The allowed number of temporary connections is configurable by the owner of the node. If a node does not menage to create a permanent connection before the temporary connection is terminated it can try again to temporarily connect to the same node as before.

3.1.5 Getting to Know New Hosts

After establishing a connection with node 1, N becomes part of the network as illustrated in figure 3.3.

It is assumed that the size of the passive - active threshold for node N is two. Node N is still going to be actively searching for nodes to connect to. N can find new nodes to connect to in two different ways.

The first way of finding new nodes is to listen for packets traversing the network. Via its connection to node 1, N is going to start receiving packets that are travelling on the network. N adds the originators of these packets into its list of known hosts. From there it fetches entries and asks these nodes if it can establish permanent connections with them. N is not going to be able to establish any more temporary connections. N does not have any need for those and they are not more over allowed for nodes that are already connected to the network.

The other way of getting to know new nodes is to query already connected nodes. For example, N can ask node 1 for its list of known hosts. Upon

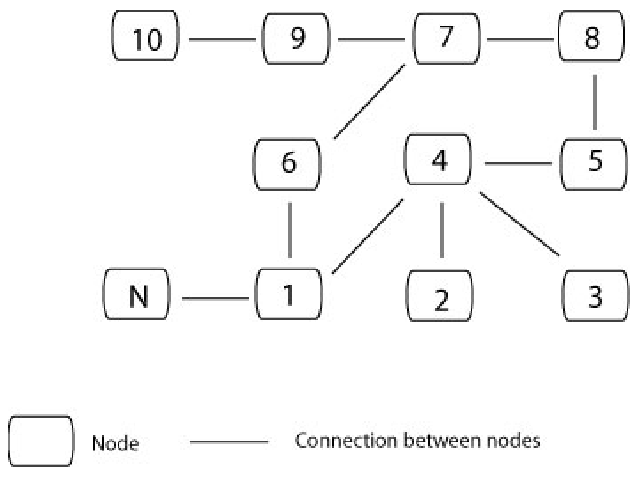


Figure 3.3: N is part of the network

receiving node 1's known hosts list, node N will add it to its own known host list and try connecting to one of these hosts.

3.1.6 Maintaining Connection

Every node on the network expects to receive packets from hosts it is permanently connected to within some given time frame. The size of this time frame can be configured manually on every host. Currently the default value is ten seconds. This might or might not be optimal. No tests have been conducted to prove that length of this time frame is optimal but it has been shown to function fine in the implementation that has been made for the purpose of this thesis. After this time frame has passed, nodes send so-called ping packets to each other. This is done to assure that nodes on the other end of the connection are still present on the network. Ping packets are not sent between nodes that are temporarily connected.

To illustrate how this work, the sample peer-to-peer network in figure 3.3 is going to be used. It is assumed that node 3 in the figure 3.3 has not received any packets from node 4 within node 3's given time frame. Node 3 is going to notice this and send a ping packet to node 4. Upon receiving the ping packet, node 4 is going to answer and reset its timer for node 3. When node 3 receives answer it is, like node 4, going to reset its timer. In case that node 4 is unreachable or does not answer, node 3 is going to assume that node 4 has left the network and drop node 4 from its list of permanent connections. Packets arriving from node 4 are not going to be accepted, nor are any new packets going to be forwarded to node 4.

3.1.7 Ending Connection

Connections on the network can be terminated in two ways. Nodes can either go down gracefully, which is preferred, or they can just vanish from the network.

There are many reasons nodes can just disappear from the network without notifying their neighbours. They can crash because of some malfunction in software or hardware, they can go down because of power failure or the system administrator could simply shut them down. When this happens, the nodes that are permanently connected to the node that has disappeared are going to notice this by not receiving any packets from that specific node. As explained in the the previous subchapter, the nodes connected to a node that has vanished are going to try to reach that node by sending ping packets to it. Because the node is gone, it is not going to send any answer. The nodes sending ping packets are not going to receive any answers and thus are going to remove that node from their list of permanently connected nodes.

When a node is leaving the network in a controlled manner it is going to send bye packet to all the nodes it is permanently connected to. The bye packet is going to contain a list of all nodes the leaving node was permanently connected to and a list of all hosts it knows. Upon receiving the bye packet the other nodes append the list of known hosts to their own list of known nodes. The receivers of bye packet also try to reach nodes the leaving node has been permanently connected to.

Contact of these nodes is not done directly with ping packets but by using the network. Ping packets that are not sent directly but forwarded on the network are called crawler pings. They are forwarded from one node to another. Among other things, crawler ping packets contain information on who is being searched, who is searching and time to live, which tells how many times the packets are to be forwarded. If they reach the node that is being searched, that node is going to answer with a ping packet. If the searched node is not reached, the issuing node is going to try to establish connections with the searched node. This is done mainly to avoid fragmentation of the network and is studied more thoroughly in example in 3.1.9.

3.1.8 Sending Request for Offers

Crawler ping packets is not the only packet type that is forwarded between nodes. Packets called request for offers (RFO) are also forwarded. Their purpose is to inform nodes that there is a job that needs to be matched to a machine. Every RFO packet contains information about the job that allows the machine that has received this packet to a involve matching algorithm and see if it satisfies the requirements set by the job issuer. Like crawler ping packets, RFO packets also contain a time to live field that terminates the packet after it has passed a certain amounts of hops.

3.1.9 Examples

Every network or system that wants to function properly all the time needs to be able to handle exceptions. In this part of the chapter several what-if cases are going to be presented. For every case there is a problem presentation and a proposed solution to that problem. As with the chapters above there is going to be an extensive use of examples and illustrations.

Fragmentation of the Network

Sometimes a network is going to consist of two subnetworks that are connected through one single host. This host is acting as a gateway between the networks. If this host leaves the network, network split is imminent. The network split creates two networks. Both of these networks are smaller and give its users less nodes to process their jobs on. Because of this, remaining nodes on the network will always try to avoid fragmentation and keep the network intact.

In a hypothetical situation, node 4 in figure 3.1 experiences a power failure and disappears off the network. Because of the nature of the failure, node 4 is not going to be able to notify the nodes it is connected to about its departure. The situation pictured in figure 3.4 is created.

As can be seen, nodes 2 and 3 are cut off from the network. They are not anymore part of the network. It is assumed that these nodes are low bandwidth nodes and have maximum connection size 1. They are after some time going to notice that node 4 is gone. They are going to send ping packets to node 4 but are not going to receive any answers. Node 4 is gone. After some time node 4 is dropped from connection tables. Nodes 2 and 3 begin

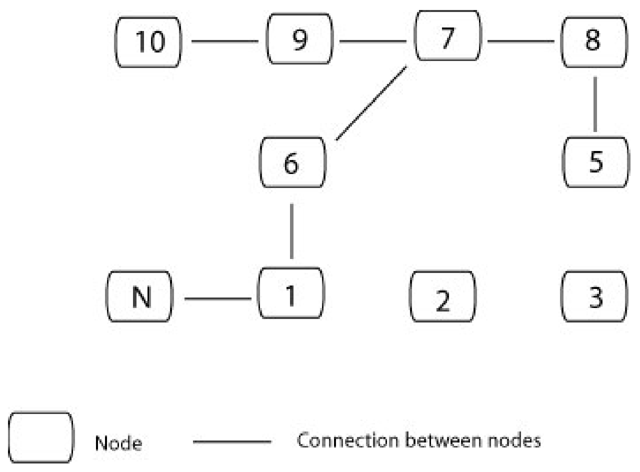


Figure 3.4: Node 4 is not part of the network

to actively search for new nodes to connect to. Hypothetically node 2 could try to establish connection with node 3 (or vice versa). This would lead to situation pictured in figure 3.5.

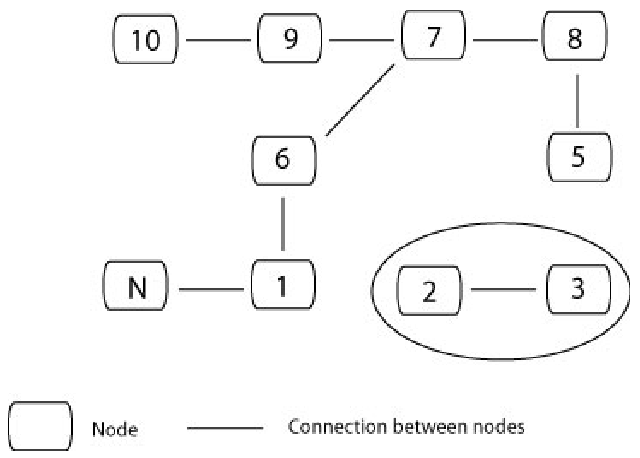


Figure 3.5: Node 2 and 3 have created their own network

As can be seen the network split has been made permanent. Node 2 and 3 do not allow more than one connection and are never going to be able to connect to the rest of the original network. There is a one network comprising of nodes 2 and 3 and the second network which contains rest of the nodes. To avoid this situation. nodes that connect to one host at most, are not allowed to connect to the other hosts that also are only allowed to connect to one host at most. It could be argued that nodes that connect to at most two hosts should neither be allowed to connect to hosts with at most one connection to avoid situation pictured in figure 3.6.

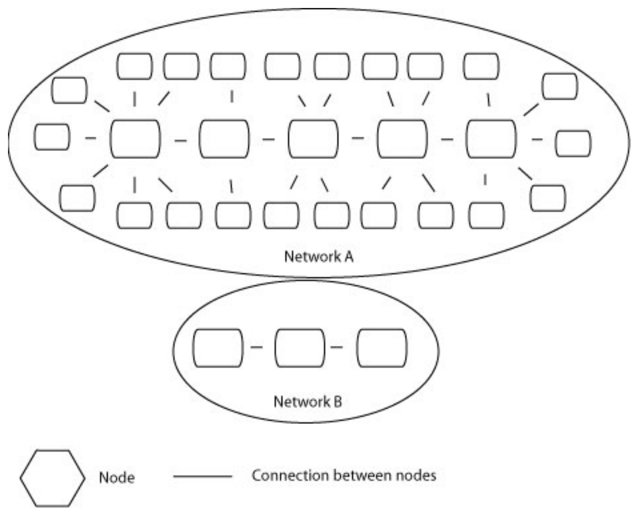


Figure 3.6: Fragmented network

Because there have not been any extensive testing of settings there can not be any arguing on what the optimal settings are. In the network implemented for purpose of this master thesis, it was only prohibited for nodes with maximum connection size one to connect to each other. That was shown to work fine but should more extensive experiments show that more prohibition is needed, it can be easily implemented.

Another case of network fragmentation can happen when hosts like node 7 in figure 3.1 goes down. Node 7 is important because it is the only node that connects the subnetwork consisting of nodes 9 and 10 to other nodes. When node 7 goes down, the situation in figure 3.7 is created. Suddenly there are two networks.

This kind of fragmentation can not be avoided if node 7 disappears without notifying its neighbours of its departure. But in the case that the host leaves network gracefully all nodes that are connected to node 7 (in this case nodes 6, 8 and 9) are going to be notified that the host 7 is leaving and what hosts it was connected to. Now these nodes issue crawler ping packet. Crawler ping packets' goal is to find out if network has been split into two

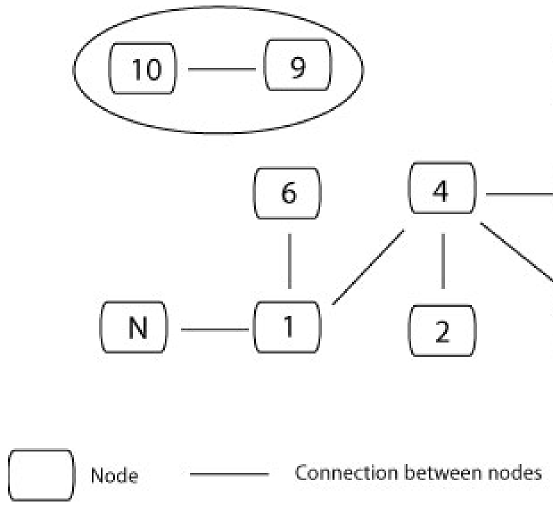


Figure 3.7: Node 7 has left the network

parts. If it is intact, nodes that node 7 has been connected to, are going to be reachable. In this case node 7 notifies nodes 6, 8 and 9 that it is parting the network. All three nodes issue two crawler ping packets. Node 8 looks for nodes 6 and 9. Node 9 is going to look for nodes 6 and 8 and node 6 for nodes 8 and 9. Now let's take a closer look at the packets issued by node 6. These packets are sent by node 6 to the nodes that it is permanently connected to, in this case node 1. After being passed through nodes 4 and 5, these crawler ping packets are going to reach node 8. Node 8 is going to notice that it is the target of the one of those two packets. Node 8 answers by sending ping packet to node 6. Now both nodes know that they can reach each other. Both are part of the same network. While the answer for crawler ping targeting node 8 has been returned, node 6 is never going to receive answer for crawler ping targeting node 9. Node 6 assumes that the network has been fragmented. After this assumption has been made node 6 tries to establish connection with node 9. Two networks have been merged into one.

Incorrect Configuration of Low or High Bandwidth Node

Sometimes users of the network are not going to be very computer literate. For example they can assume that their low bandwidth node can easily handle thirty connections to other nodes. Some other time they might decide that their high bandwidth node should not connect to more than one or two other nodes. So how do those erroneous decisions affect the network?

In case of erroneous configuration of a low bandwidth node, the performance of network is not going to be seriously hurt. Nodes 2 and 3 are again assumed to be low bandwidth nodes on sample network from figure 3.1. The difference from previous examples is that the owner of node 2 decides that he should set the threshold between passive - active searching to thirty. Thus node 2 strives to connect to at least thirty other nodes. For every extra host node 2 connects to, the amount of traffic flowing from and into the node increases. After some time the network link on node 2 is going to get saturated. Node 2's connection to outside world is clogged. Packets are going to be dropped and the owner of the node is not going to be able to do anything meaningful with his or her machine. Because other nodes are configured properly only node 2 is going to be hurt by this faulty configuration.

On the other end of the scale is the erroneous configuration of high bandwidth nodes. Instead of connecting to ten or maybe fifteen other nodes, the high bandwidth host connects to only one host. The owner does not understand benefits of being connected to many nodes. Like in the case with faulty configuration of low bandwidth node, this is not either going to decrease performance of the network as long as there are not many users who configure their high bandwidth nodes incorrectly. If there are many users who do this, network performance is probably going to degrade because of lack of nodes acting as the backbone of the network.

Keeping Resources for Itself

As with any other system with several users, there are going to be many misbehaving users. Sometimes some specific resource on the network is going to be a high performance computing node. It can be a powerful vector computer or a cluster of Linux nodes. As such it is going to be highly sought after. If it is assumed that node 3 in figure 3.1 is such a high performance node and that the owner of node 4 knows this, a problem could arise. As can be seen in figure 3.1, node 4 is node 3's only link to the rest of the network. A Malicious owner of node 4 could configure his node not to forward any job requests to node 3. The owner of node 4 is keeping node 3 for his own computing needs. To avoid this, nodes on the network need to be able to analyse job requests. This will allow all nodes on the network to notice offending nodes and disconnect those. It is very unlikely that this problem is going to arise as high performance computing nodes are probably going to have a fast connection to the outside world and be able to connect to ten - fifteen hosts instead of just one. Even if this is not likely to happen it is important that the network can handle situation like this no matter how unlikely it is.

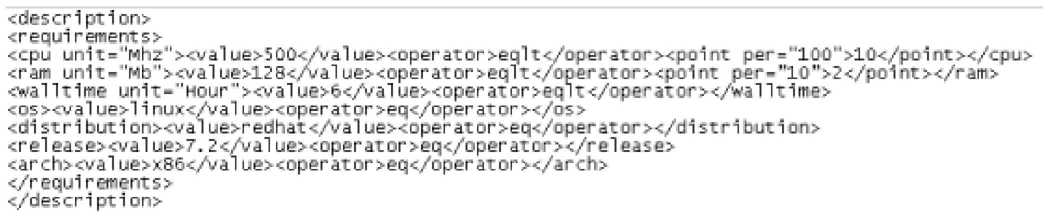
3.2 Brokering

Every machine which is participating as a node on the network contains a description file. This file describes what resources a specific machine has to offer to different jobs and what requirements it poses on jobs that want to execute on that specific machine. For example how fast the clock speed of CPU is or the amount of Random Access Memory (RAM). It is not necessary for a description file to provide true values. A specific machine might have 512MB of RAM, but the owner wants only to allow jobs that do not require more than 256MB of RAM. Thus the owner specifies 256MB of RAM in the machine description file. As every machine's requirements are described by a machine description file so is every job submitted to the network described by a job description file. This file is similar to machine description files but describes job properties and requirements instead of machine requirements. The following chapter contains a description of machine and job description files. It also contains a description of how a user submits jobs to the system.

3.2.1 Job and Machine Descriptions

Before submitting a job, a job specification needs to be written. The job specification is written as a job description file. This file needs to follow the rules defined in a W3C XML Schema [16], which can be found in appendix A. This XML Schema contains all the rules and requirements necessary and should be as such studied carefully by the network users. As it is a plain text file, job descriptions can be written in simple text editor, for example jed on UNIX systems or notepad on Windows. The machine description file is also a plain text file and follows exact the same rules as job description file.

Before becoming part of the network every node reads its own machine description file. Thus machine description files need either to be written by the user and made available for node on upstart or the node needs to be able to check the machine's properties on upstart. This checking of machine properties is done in different ways on different operating systems. For example on Linux machines the node could parse /proc directory, gather necessary information from there and create machine description file automatically. A machine description file can look like this:



```
<description>
<requirements>
<cpu unit="Mhz"><value>500</value><operator>eqlt</operator><point per="100">10</point></cpu>
<ram unit="Mb"><value>128</value><operator>eqlt</operator><point per="10">2</point></ram>
<walltime unit="Hour"><value>6</value><operator>eqlt</operator></walltime>
<os><value>linux</value><operator>eq</operator></os>
<distribution><value>redhat</value><operator>eq</operator></distribution>
<release><value>7.2</value><operator>eq</operator></release>
<arch><value>x86</value><operator>eq</operator></arch>
</requirements>
</description>
```

Figure 3.8: Machine description file

3.2.2 Rules

As was mentioned earlier, the XML Schema in appendix A defines rules for job and machine description files. The syntax of machine and job description files is heavily influenced by previous work in this area, specifically Condor's ClassAd mechanism [1] [15], which is briefly described in chapter 2 and SSE draft. Upon reading the machine or job description file the node is going to check validity of these files against schema in appendix A. The validation of the file is done at the same time as the parsing of a description file. Node's machine description file together with job description files provides necessary requirements for making matching of jobs and machines possible. The job specification contains all the information that the nodes on the network are going to need to successfully execute the job should they choose to accept that specific job.

Job description files consist of two parts. The requirement part specifies job requirements and preferences, while the environment part specifies all the information needed for successful execution of the job. While the job specification consist of two parts, machine description files do not need the environment part and consist only of the requirement part.

The requirement part contains machine and job requirements and preferences. Everything placed between <requirement>and </requirement>is considered as part of the requirement part. As can be seen from figure 3.8 the requirement consists of several nested tags that in turn also consist of several other nested tags. These tags are defined in the XML Schema in appendix A. Every valid tag in the requirement part must contain two tags inside them and can optionally contain one more tag. The tags "value" and "operator" are obligatory while the tag point is optional. Here a follows short explanation of these three tags.

The value tag is as previously mentioned obligatory and nested inside every tag that is part of the requirement. In case the tag value contains a non-string value, the tag is required to have the attribute "unit". This attribute allows different users to use different units when specifying job and machine properties. For example a node owner can express his machine CPU clock speed in MHz while the specification for job CPU clock can be written in GHz. Thus 0.5 GHz and 500 MHz represent same value. The conversation of different units is done automatically by the system.

The "operator" tag is also an obligatory part of all the tags that are part of the requirement. Its value can be one of following six

- equal
- not equal
- less than
- equal or less than
- greater than
- equal or greater than

With the help of these six operators, a job can be matched to a machine.

Lastly, tags in the requirement can contain the tag "point". With the help of the "point" tag the job issuer can express his or her preferences. For example the user can state that his job can be executed on machines with 128 Megabyte or more of RAM. For every additional 10 Megabytes of RAM that the offering machine has to offer, it is to be "awarded" some amount of points. In the following example:

```
<cpu>
  <value unit="MHz">500</value>
  <operator>eqmt</operator>
  <point per="100">10</point>
<\cpu>
<ram>
```

```
    <value unit="Mb">128</value>
    <operator>eqmt</operator>
    <point per="128">2</point>
  <\ram>
```

job specification awards offering machine with 2 points for every additional 10 Megabytes of RAM. In the same example every additional 100 MHz of CPU clock speed are awarded 10 points. If two machines, the first one with following preferences:

```
  <cpu>
    <value>600</value>
    <operator>eqlt</operator>
  </cpu>
  <ram>
    <value>138</value>
    <operator>eqlt</operator>
  </ram>
```

and the second one with these preferences:

```
  <cpu>
    <value>800</value>
    <operator>eqlt</operator>
  </cpu>
  <ram>
    <value>128</value>
    <operator>eqlt</operator>
  </ram>
```

answer a job request, the job issuer is going to compare these two machines according to his preferences. The first machine is going to receive 12 points while the second one is going to receive 30 points. Machine number two has more points than machine number one. Thus machine number 2 suits the job issuer better and is chosen to process the job.

Environment tags denote the part of the job specification that contains information on the environment needed for the successful execution of the job. Inside the environment part three different tags can occur

- download
- upload
- execute

The "download" tag specifies the path of a file or directory that should be downloaded and the name of that file or directory on the system after it has been downloaded. For example

```
<download>
  <name>localName</name>
  <path>ftp://example.com/someFile</path>
</download>
```

specifies that file someFile on FTP host example.com should be downloaded and then renamed to localName after it has been downloaded. The environment part can contain several download tags. In that case all files are downloaded. Downloading of files can be done in different ways. In the above example the file was to be downloaded from an FTP server. Nodes can also use different protocols as HTTP or TFTP to download files.

The "execute" tag can only occur once in a specific environment tag. It specifies which file should be executed after downloading all files. Usually, a file to be executed is a simple shell script that initializes all the variables required by the job and then starts the job. Usually the "execute" tag looks like this:

```
<execute>localFileName</execute>
```

Finally after finishing the job, node looks at the upload tags. Upload tags have similar syntax as download tags. They consist of the name of a local file and the path to which the file should be uploaded. For example

```
<upload>
  <name>localFile</name>
  <path>ftp://example.com/remoteFile</path>
</upload>
```

specifies that the node should connect to the FTP server example.com and store the file localFile there as remoteFile. As with download, different protocols can be used for uploading and several files can be uploaded by using multiple upload statements.

After writing the job description file, the user is ready to submit a job. The user submits a job from his or her local node (henceforth L). L parses the job description file and creates a request for offer (RFO) packet. Besides containing information from the job description file, RFO packets have four more fields. A Unique packet identification number (UID), a Time to live (TTL) field, contact information for node issuing job request and contact information of node the forwarding this packet. After node L has created an RFO packet it sends it to all of its neighbouring nodes.

Upon reception of this packet node N that is neighbour to L examines the packet UID. This is done to avoid processing packets that have already been processed once. It also prevents loops in the network, where the same packet is sent around several times. In the case that the packet UID matches UID of some other packet that has already been received, N is going to discard the packet. Otherwise the packet is kept.

After checking the packet UID, N examines the TTL field. This field is usually initiated to between seven and ten by the job issuer (in this case L). Every node that receives the packet decreases this field value by one. Thus L decreases this packet by one also. In the case that TTL field value is zero or less the packet is discarded. As can be seen in figure 3.9, the size of the TTL field decides how many hosts are going to be reached by the request.

If the packet has passed both of these aforementioned checks, node N is going to pass the packet to its neighbours. The packet is not sent to L because L is the node that sent the message to N and thus has this RFO packet already.

After sending the packet, node N is going to check its own status. There are three possible states for every node on the network. These are

- available
- matching
- working

Available means that the node is not doing any work and can accept jobs should they match the node's preferences. Matching means that the node is

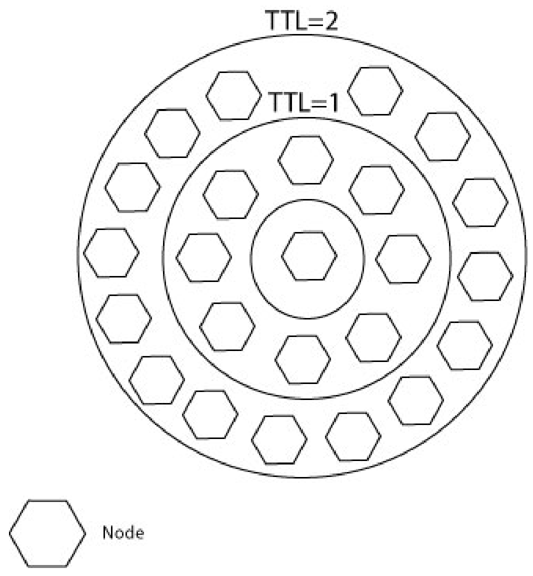


Figure 3.9: High TTL increases range of the packet

currently invoking the matching algorithm or has matched a job to itself and is waiting for the job issuer to answer. In the case that the job issuer does not answer in some pre-configured time frame, the state of the node is going to be reverted back to available. When the node starts to process the job its state is changed to "working". If while checking its status, node N finds out that the status is "working" or "matching" it is not going to do anything. If status is "available" node N is going to invoke the matching algorithm.

The matching algorithm is a piece of code that decides if both machine and job requirements can be met. The match is said to be successful when both machine and job requirements have been met. The matching of job to specific node is done in two steps. First, a node that has received job request in form of the RFO packets checks if both machine's and job's requirements are satisfied. If both have been satisfied, the node answers to the job issuer. Now the job issuer applies the matching algorithm again to verify the job applicant's claims. If they are verified, the job issuer stores information about the successful job application and the number of points the job applicant received. The job issuer does not immediately answer to first successful job applicant. Instead it waits for some pre-configured amount of time so that nodes that made a successful match have enough time to answer. This allows even nodes that are located far away (on the network) to be treated fairly.

After this time frame has passed, the job applicant that got the highest value according to the point mechanism is chosen to process the job. The process in which a specific node switches between these three above mentioned states is illustrated in figure 3.10.

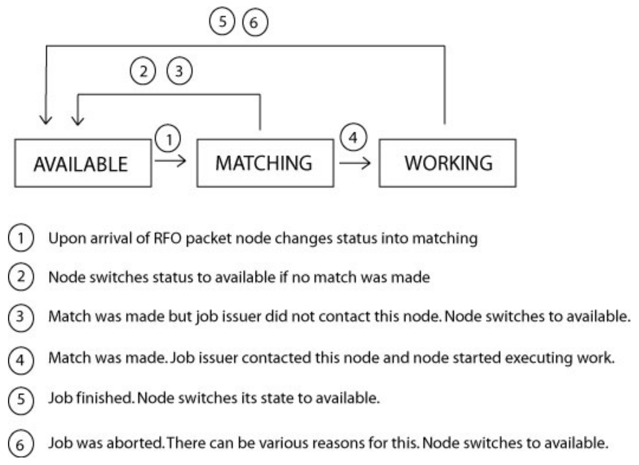

5 6
2 3
AVAILABLE
1
MATCHING
4
WORKING
1 Upon arrival of RFO packet node changes status into matching
2 Node switches status to available if no match was made
3 Match was made but job issuer did not contact this node. Node switches to available.
4 Match was made. Job issuer contacted this node and node started executing work.
5 Job finished. Node switches its state to available.
6 Job was aborted. There can be various reasons for this. Node switches to available.

Figure 3.10: Rules for switching between different node states

The matching algorithm on nodes that are issuing the job and on nodes that are applying for a job works in a similar fashion. Job description files allows the user to define name of the specific resource that is requested. It also allows them to define how much of that resource is needed. For example the following job description file:

```
<os>
  <value>linux</value>
  <operator>eq</operator>
</os>
<cpu>
  <value unit="MHz">1500</value>
  <operator>eqmt</operator>
</cpu>
```

requests that the operating system on node that is applying for the job must be Linux and that the CPU clock speed should at least be 1500Mhz. The user who has written example above, has made clear that a match can only be made if the job applicants have values for CPU clock speed and operating system, that falls into intervals 1500 or more (illustrated in figure 3.11) and Linux (illustrated in figure 3.12).

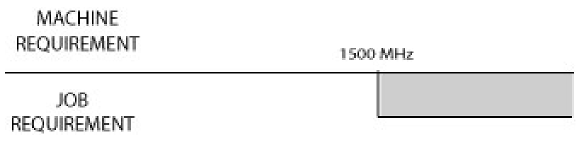


Figure 3.11: Only machines with CPU clock speeds over 1.5 GHz are accepted

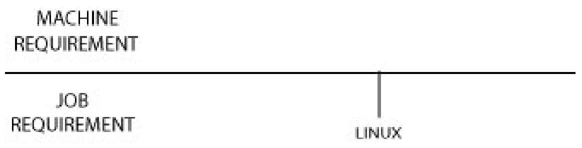


Figure 3.12: Only machines running Linux are accepted

The nterval in figure 3.13 is infinitely small and is only matched by the value Linux. Node A has received an RFO packet containing requirements from above mentioned example. Node A's machine description file looks like this:

```
<os>
  <value>linux</value>
  <operator>eq</operator>
</os>
<cpu>
  <value unit="GHz">2</value>
  <operator>eqlt</operator>
</cpu>
```

This means that node A is accepting jobs in the intervals illustrated in figure 3.13 and figure 3.14.

The matching algorithm is going to compare the intervals allowed by the job issuer and the job applicant. Figure 3.15 and figure 3.16 illustrate this.

If there is overlapping between the intervals it means that the match is successful. As can be seen there is indeed overlapping and thus a match is made. Now the matching algorithm can count the amount of points that the applicant has been awarded. As there was no ”point” tag in the job

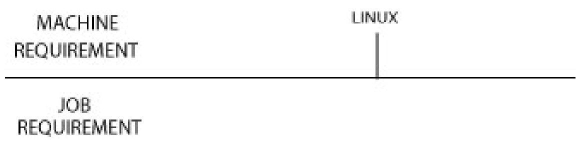


Figure 3.13: Only jobs which require Linux are matched to this machine

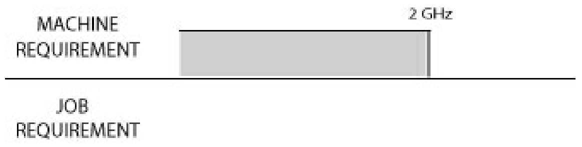


Figure 3.14: Only jobs requiring 2Ghz or less are matched

application, the job applicant receives only one point. If the job application looked instead like following example:

```
<cpu>
  <value unit="MHz">1500</value>
  <operator>eqmt</operator>
  <point per="100">2</point>
</cpu>
```

the job applicant would have been awarded 10 points. For every 100 megahertz over 1500 megahertz it would receive 2 points, thus 10 points. If the job applicant's machine description file looked like the following example:

```
<cpu>
  <value unit="GHz">1</value>
  <operator>eqlt</operator>
<cpu>
```

the matching would be unsuccessful. As can be seen in figure 3.17 there are no overlapping intervals and thus no match.

As has probably been noticed, the matching algorithm assumes AND statements between tags with different names. All requirements need to be satisfied for a successful match. But what happens if there are two tags with the same name? Which tag is the right one? In this case both tags are the right one. The matching algorithm assumes OR statement between tags with the same name. One of the requirements needs to be satisfied for successful

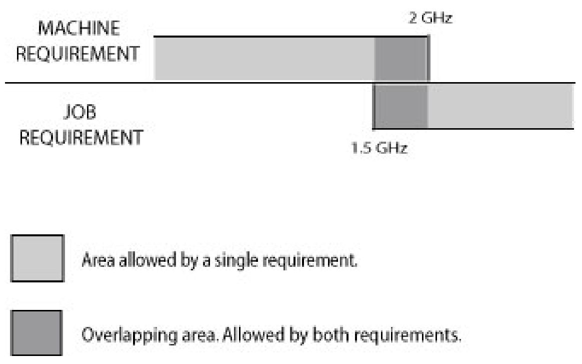

MACHINE REQUIREMENT
2 GHz
JOB REQUIREMENT
1.5 GHz
Area allowed by a single requirement.
Overlapping area. Allowed by both requirements.

Figure 3.15: Overlapping means that the match is possible

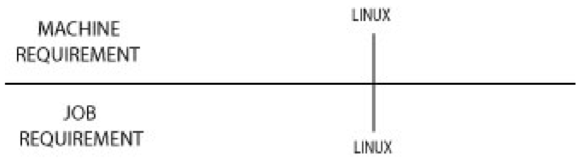


Figure 3.16: Machine is running OS that the job is requsting

match. This feature is useful when a job for example can be executed on two different operating systems. For example

```
<os>
  <value>linux</value>
  <operator>eq</operator>
</os>
<os>
  <value>netbsd</value>
  <operator>eq</operator>
</os>
```

means that job runs on Linux or NetBSD operating system. Thus machines that are running either Linux or NetBSD are going to match to this job. This is illustrated in figure 3.18.

Appendix B has a table over all possible outcomes of matching when six different operators are used in machine and job description files.

After the job applicant receives confirmation that it has been chosen for processing the job, it looks at the environment part of the job description

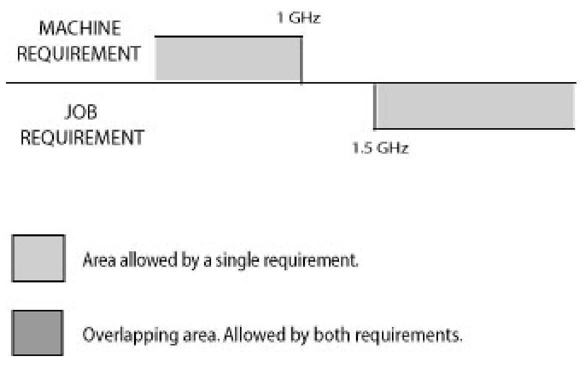

MACHINE REQUIREMENT
1 GHz
JOB REQUIREMENT
1.5 GHz
Area allowed by a single requirement.
Overlapping area. Allowed by both requirements.

Figure 3.17: There is no overlapping and thus no match

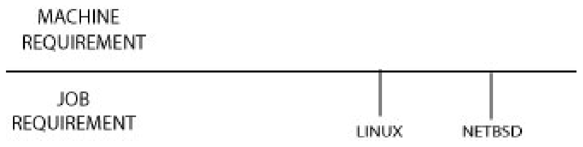


Figure 3.18: Machine matching this job should run one of these two OS

file. As mentioned previously, the job applicant will now download all files that are marked for downloading by the job description file and then execute one of them as noted by the same job description file. The status of the job applicant is going to change from "matching" to "working". After the job has been processed and files have been uploaded, the status changes from "working" to "available" and the node can bid for jobs again. Once again, for better visualisation of the process of switching states the reader is referred to figure 3.10

3.3 Security

This subchapter concerns itself with the security of the network. For the purpose of understanding the network's security model there is a short explanation of basic cryptography and concepts like digital signatures and certificates. This explanation is not intended as a complete description of abovementioned concepts but more as a rehash of the reader's existing knowledge. For a more thorough description the reader should check Code Book [18] and PGP: Pretty Good Privacy [8]. This chapter also examines threats to the network explained above and gives possible solutions to these threats.

3.3.1 Different Cryptographic Concepts

Since the beginning of the times humans have had a need of communicating with each other without a fear of someone overhearing their conversation. With the help of cryptography ordinary people, big companies and national states can communicate without the fear of being eavesdropped. The use of the encryption can be traced as long back as early Egypt and Mesopotamia [18]. It relies on a branch of mathematics known as cryptography and allows two parties to communicate securely with each other. There are two main groups of ciphers, symmetric and asymmetric. Here follows a short explanation of these.

The basic concept of all symmetric ciphers is that the message text, known as plaintext gets encrypted with the help of a secret key. The encrypted message is referred to as ciphertext. To extract the plaintext from the ciphertext the user would need to run the encryption algorithm backwards with the exact same key that was used for encryption of the message. This procedure is illustrated in figure 3.19.

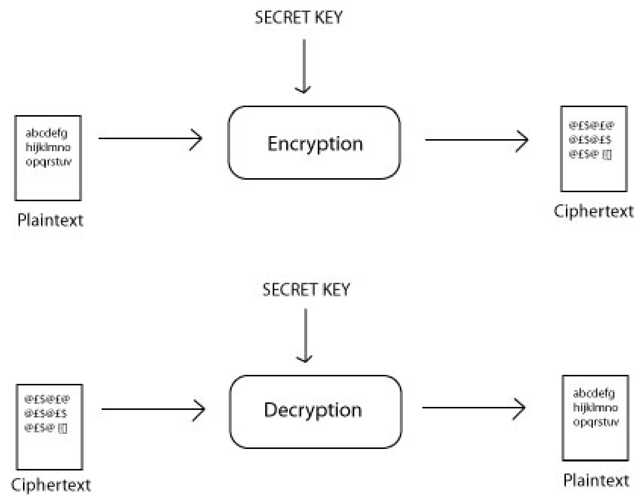
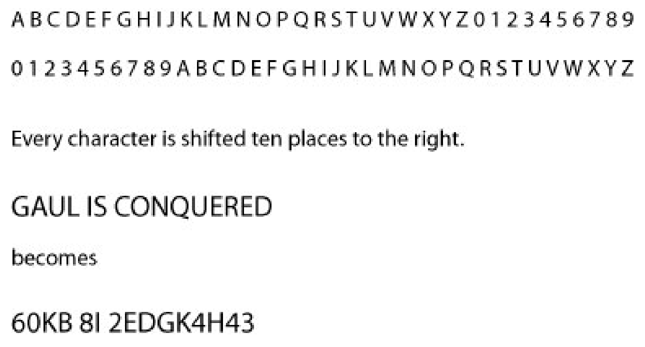


Figure 3.19: Symmetric encryption and decryption

The oldest forms of ciphers are substitution ciphers. As was mentioned earlier they were used as long back as old Egypt. To encode messages with the substitution ciphers user would have to replace individual letters with other symbols. For example the user could, as Julius Caesar did with his secret code [8], simply shift the alphabet three places to the right. After doing this, the message would be encoded. Figure 3.20 pictures a message encrypted with a variant of Caesar's cipher.


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Every character is shifted ten places to the right.

GAUL IS CONQUERED

becomes

60KB 8I 2EDGK4H43

Figure 3.20: Example of how the Caesar cipher can be used

Other commonly known substitution ciphers are the Freemason cipher and the George Washington cipher. For every year that passed ciphers evolved more and more. Big leaps in cryptography were especially made during World War II, when the need for securely transmitting messages was bigger than ever. For more thorough history of encryption reader should check The Code Book [18]. Today the most used symmetric ciphers are Triple-DES (Triple Digital Encryption Standard) and AES (Advanced Encryption Standard).

Cryptography with the symmetric ciphers presents user with several problems. For every communication between two users there is a need for a secret key which is known by both users. When three users want to communicate with each other there is going to be a need for three different keys. One for every channel of communication. This is illustrated in figure 3.21.

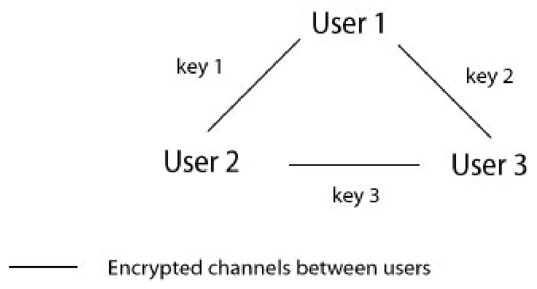


Figure 3.21: Three entities communicating with each others

If one more user wants to join communication, like in figure 3.22, the need for number of secret keys grows substantially.

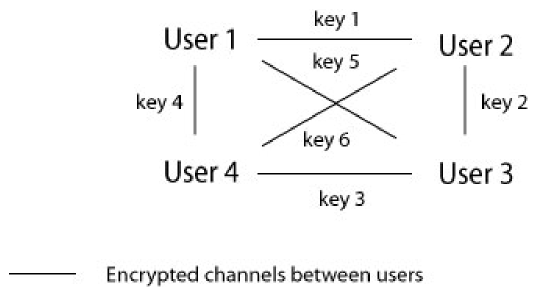


Figure 3.22: Four entities communicating with each others

The number of keys needed for secure communication does not increase linearly but with:

$$\frac{n\times(n-1)}{2}$$

One, even bigger problem, is that to be able to start secure communication between two parties there needs to be an exchange of secret key prior to this start. Secure communication without prior arrangement is impossible. And to exchange a secret key is not always easy. One user could be located in Sweden while the other one may be located in Cuba. To exchange a secret key they would need to have some secure line of communication. But if they had one there would not be any need for exchanging a secret key. They could simply use the existing secure line for communication.

This problem remained unsolved until 1976. Asymmetric ciphers or public key cryptography was born that year. In public key systems there are two keys. The private key is used for decrypting messages while the public key is used for encrypting messages. This is illustrated in figure 3.23.

The two users from previous example, one in Sweden and another one in Cuba can now communicate securely. User A in Sweden publishes his public key on the Internet. A does not need to keep his public key secret. User B in Cuba (or anyone else who wants to communicate with A) encrypts their message with A's freely available public key. After reception of the message, A uses his or her secret, private key to decrypt the message.

Another important aspect of public key cryptography is its ability to sign documents. Sometimes the user does not need to keep his documents out

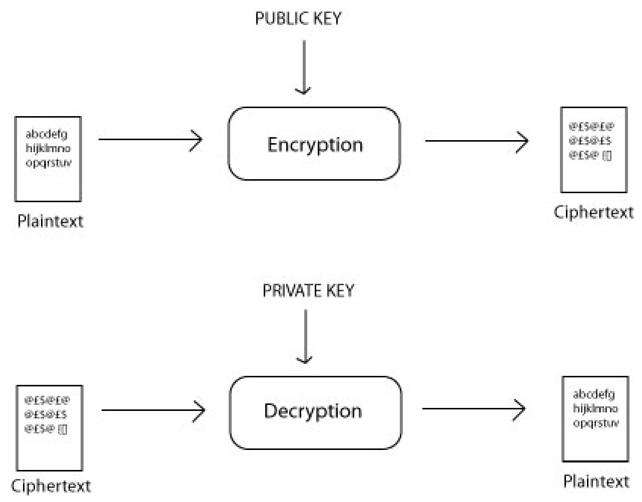


Figure 3.23: Encryption and decryption with assymetric keys

of the public eye. He might only want to keep malicious users from altering them. Digital signatures can be used to keep documents from being altered and to verify the document author.

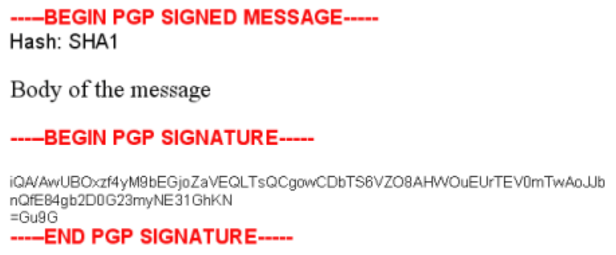
How does a digital signature work? In a simple model user A could sign message with his private key and thus make ciphertext out of it. Then everyone else could decrypt the message with the help of the user's publicly available key. If other users can decrypt the message with user A's public key they are going to know that user A really encrypted this message. Thus the originator of the document is verified.

But what if a user wants to keep the message unencrypted but still wants to prevent other users from altering the document? Systems like Pretty Good Privacy (PGP) solve this problem by first processing the message through a message digest function. This function returns a value that correspons to the message. This number is then signed with the user's private key and is appended at the bottom of the message. Figure 3.24 contains a snippet of a PGP signed message.

Hash functions (otherwise known as message digest functions) provide a fingerprint of the data. Sending message M into function H generates fingerprint f.

$$F = H(M)$$

To provide a fingerprint of the message, a mathematical function must



```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Body of the message

-----BEGIN PGP SIGNATURE-----

iQA/AwUBOxzf4yM9bEGjoZaVEQLTsQCgowCDbTS6VZO8AHWOuEUrTEV0mTwAoJJb
nQfE84gb2D0G23myNE31GhKN
=Gu9G
-----END PGP SIGNATURE-----
```

Figure 3.24: Snippet of PGP message

have [19] the following properties:

- The function H can be applied to message of any size
- H produces fixed length output
- $H(x)$ is easy to compute
- For any given value f, it is computationally infeasible to find x such that $H(x) = f$
- For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$
- It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$

Together these properties of a specific mathematical function allow for secure fingerprinting of the message.

3.3.2 Threats and Suggested Solutions

According to the Internet Security Glossary [17], a threat is a potential for violation of security, which exists when there is a circumstance, capability, action or event that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability. ITU-T (The International Telecommunication Union, Telecommunication Standardisation Sector) has defined the recommendation known as X.800 Security Architecture for OSI [20] that defines requirements for security and ways to satisfy those requirements. With the help of the above mentioned documents this chapter is going to examine threats posed against the network described in previous chapters and possible solution to these threats.

X.800 defines five security services, which a system should provide to give a specific kind of protection to system resources. These five services are authentication, access control, data confidentiality, data integrity and non-repudiation.

Authentication

The authentication service should provide assurance that communicating entities are the ones they claim to be. At the initiation of a connection, the authentication service assures that the two nodes are who they claim to be. In the transfer phase it provides assurance that originator of data is as claimed. This service also provides protection against interfering of a third party that may masquerade as one of the two communicating entities.

The authentication service does not need to be fully implemented in the network described in previous chapters. The reason for that is that the very nature of peer-to-peer networks makes the first part of this service obsolete. Every node on a peer-to-peer network is equal. Thus there is no need for a node to impersonate some other node upon connection to the network. An impersonating node does not receive any benefits from this. Nodes already on the network treat all incoming nodes equally. Thus there is no need for trustful identification of nodes upon their connection to the network. Nodes are assumed to be what they claim to be.

But there is a need for the second part of the authentication service. Some nodes on the network are going to be configured only to accept jobs from certain users. For example node A may be configured only to process jobs from user B and ignore jobs from user C. Malicious user C may try to act as user B and gain computing resources available at A. Thus there is clearly a need for providing assurance that a message originator is who it claims to be.

The solution to this problem is closely related to the solution of data integrity problem. Because of that solution is presented later together with the solution of the data integrity problem.

Access Control

The goal of the access control service is to prevent unauthorised use of resources. For example this resource defines what those who are allowed to use node may use it for. User B might be allowed to use node A for execution of jobs, but he or she is not allowed to overwrite node A's system directories. The enforcement of these rules is access control services' goal.

The nodes on the network rely on the underlying operating system for enforcement of access control. Jobs spawned by the nodes are always owned by a user with limited access rights. This user is usually called ”node” and is created upon installation of a specific node. His rights are controlled and enforced by mechanisms in the operating system that the node is running on.

Data Confidentiality

The purpose of the data confidentiality service is to protect data from unauthorised disclosure. User B sending his job to node A should be able to trust that user C is not going to be able to get hold of his data.

This problem can be divided into two parts. The first part concerns protection of data on hard drive. Data is going to be stored both at node A's protected areas and user B's private area. It must be protected both by node A and user B. Protection of this data is assumed to be handled by the operating system. Even if malicious user C is located at same computer as user B, he or she should not be able to get hold of data in user B's private area. The second part of the problem is how to protect data while transferring it between A and B. The solution to this is to use passwords and encryption of data while transferring it. For example instead of using an unencrypted file transfer protocol (FTP) for the transportation of files, A and B can use secure copy (scp) which encrypts all communication between endpoints.

Data Integrity

The data integrity service assures the system that data received does not contain any modification created by a third party on the way between the source and the target node. It also assures that no messages are replayed, inserted or deleted by third part.

The solution to the modification, insertion and even to the authentication problems from chapter 3.3.2 is to use digital signatures and hash functions.

Node A needs to have access to user B's public key. A obtains it from either B or some publicly available repository of public keys. At the other end user B puts the message into a hash function and receives a fingerprint of the message. Later B encrypts this fingerprint with his or her private key and sends the message together with the fingerprint to the network. At node A, the message is going to claim that the originator is B. A tries to decrypt the encrypted fingerprint of the message. Decryption is done with B's public key. If decryption is successful, A has verified that user B is indeed the source of the message. After the source has been verified, A is going to involve the same hash function that B used to produce fingerprint of the message. If the received fingerprint is the same as the one received in the message it means that the message has not been altered. If the received fingerprint is different, A is going to realise that the message has been altered. Altered messages are discarded. Because of its great role in the network's authentication service both public-private key system and hash function used should be chosen with great care.

Now that the problems of modification and insertion of messages have been solved, we can take a look at the solution of message replay and deletion problem. In one scenario user C could try to gain access to computing resources at node A by replaying messages that node A received from B before. How is node A to know that messages are replayed? If there were an easy way to synchronize clocks around the network, addition of time stamps to all the messages would solve this problem. However there can not be any assumption of time synchronization between clocks on the network and thus this solution is invalid. One way of solving this problem is to use one-time passwords for retrieving files. For example user B stores files to be downloaded at specific FTP sites. B writes the location of the files in the job description file and informs A. A logs into the FTP site with the one-time password that was received from node B. The files are downloaded and A begins processing the job. What A and B did not know is that communication between them was recorded by user C. Some time later C takes control of B's FTP site and replaces B's files on FTP server with his own. After that C starts replaying messages for A. A notices that it is instructed to download files from exact same FTP server and with exact same password as before. Because all passwords for downloading are one-time passwords, A refuses to download and process jobs.

Introducing a field with sequence of numbers solves deletion problem. Every time A sends a packet to B, this field's size is increased by one. For example if the value of this field is 10 when the message is sent to B, A expects that the next packet from B should have field size 11. If the message is deleted A is not going to receive a message with field size 11. In that case, A can ask B to resend the message with field size 11.

Non-repudiation

The non-repudiation problem is the denial of one of two parts in communications, that they have been involved in this communication. The non-repudiation problem is divided into two parts. Denial of origin of the message and denial of destination of message. In the first part, sender of the message denies that it has sent the message. In the second part, receiver of the message denies that the message has been received.

In the peer-to-peer network described in previous chapters this problem can be manifested in two ways. First, user B can deny that he or she has ever sent jobs to node A. In this case the work done by A is judged worthless. But because node A would spend time as part of the network anyhow there is no great loss. Because of not being a great loss, this problem is not handled in the network. The second manifestation of the problem is when node A denies that it has done work for user B. This problem is expected to be handled by the user of the network. It is up to the user who has received faulty results from node A to decide if he is going to continue trusting the owner of node A. If not, the user chooses simply not to allow any more jobs to be processed by nodes owned by the same user as node A was.

Chapter 4

Implementation

For the implementation of the system described in this report programming language Python was used. Python is an object-oriented scripting language. It is very powerful, free, easy to use and portable. Perhaps the most important reason for choosing Python was that it allows for rapid prototyping. As a part of the standard Python distribution a library known as XML-RPC follows. This library was used extensively for programming network interface. All communication between nodes on the network is done with the help of XML-RPC.

XML-RPC is library built for distributed computing. It is a specification and a set of implementations that allows programs to communicate with each other. It does not matter in what environment the programs work. Programs can be implemented in any language and be run on any system.

At the server side, the programmer creates a function that is then registered. After registering this function, it can be reached by the clients. Now the client only need to contact the server and call that function. For this remote procedure calling, HTTP is used for the transport while the XML is used for the encoding of the data. The library is very powerful but is still very easy to use.

In the previous chapter it was explained that the system consisted of several layers. These layers were communication, brokering and security/execution. The code of the programme also mimics this layer layout. Functions from different layers are part of different files. This is illustrated in figure 4.1.

Because of the code length there will not be any thorough examination of the code here. The most interesting part of the code is probably the part of the code used for parsing the job and the machine description files. As

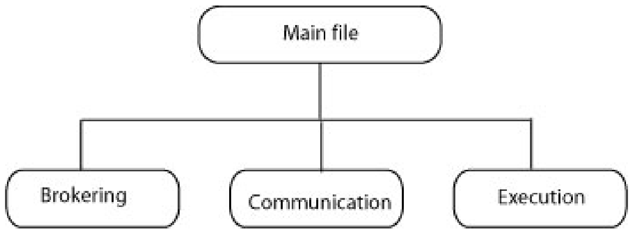


Figure 4.1: Code overview

it is already said description files are proper XML documents. They follow rules created by the XML Scheme in the appendix B. Parsing of XML files can be done in several different ways. Currently The Simple API for XML (SAX) and World Wide Web Consortium (W3C) recommendation DOM are the most popular methods for working with XML Data. DOM is designed to be a language-neutral interface to an in-memory representation of an XML document and is the method that is used in the program. In DOM, every piece of XML data is a node represented by a node object. Upon loading of the program, every node parses its own machine description file. A tree of nodes like the one illustrated in the figure 4.2 is created.

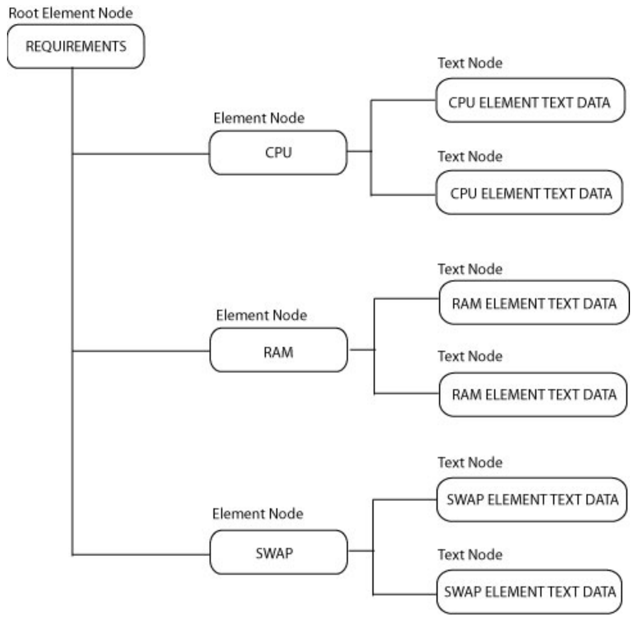


Figure 4.2: DOM Data tree

After the tree has been created data needs to be fetched from it. This is done by traversing the tree.

4.1 Example

This subchapter is going to present a submission of a work unit on hypothetical network through the use of the screenshots from different nodes that are present on the network. This hypothetical network is presented in figure 4.3.

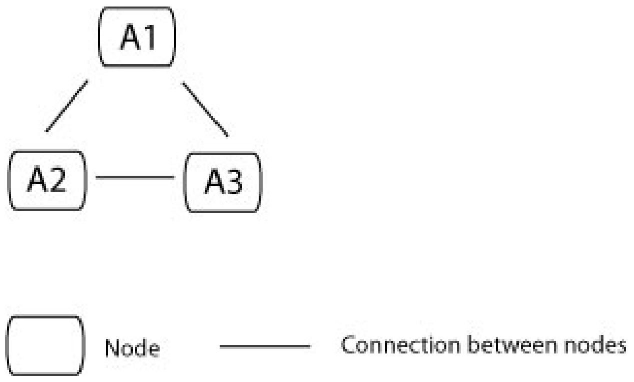
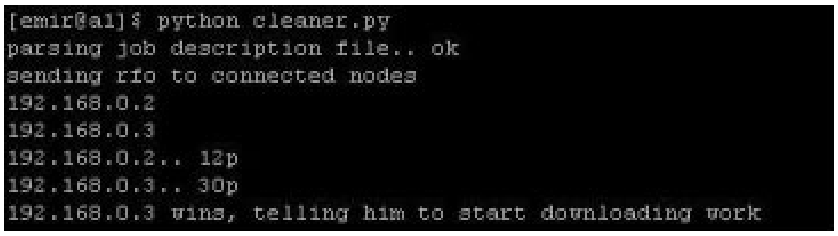


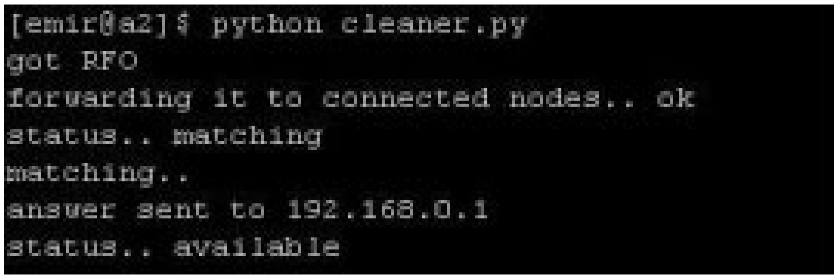
Figure 4.3: Network consisting of three nodes

It consists of three nodes that are connected to each other. The nodes are named A1, A2 and A3. A1 submits the job in the form of a RFO packet. The nodes A2 and A3 receive this packet and invoke the matching algorithm. Both nodes send back positive answers to the node A1. The matching node A1 realizes that the node A3 fits its job better and thus it chooses the node A3 to process the job. A3 switches its state to "working" and downloads the work and starts processing it. At the same time node A2 reverts its status to "available". After it has finished processing the work A3 uploads processed work and changes its status to "available". Figure 4.4 contains screenshot from node A1, figure 4.5 contains screenshot from node A2 and finally figure 4.6 contains screenshot from node A3.



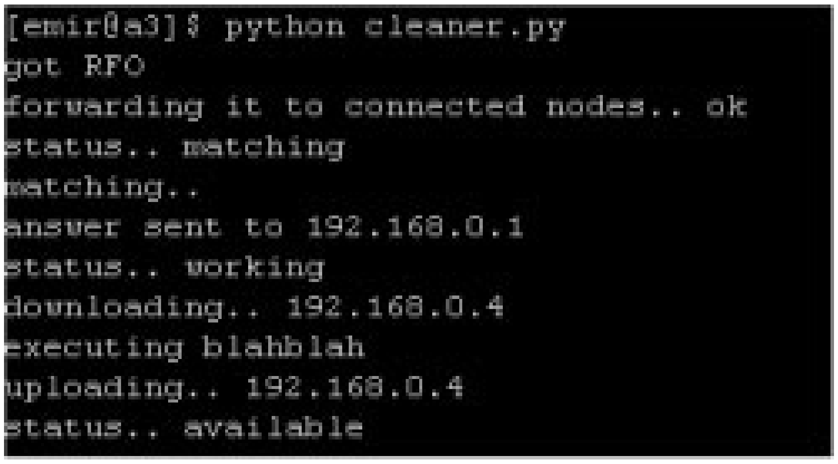
```
[emir@a1]$ python cleaner.py
parsing job description file.. ok
sending rfo to connected nodes
192.168.0.2
192.168.0.3
192.168.0.2.. 12p
192.168.0.3.. 30p
192.168.0.3 wins, telling him to start downloading work
```

Figure 4.4: Screenshot of node A1



```
[emir@a2]$ python cleaner.py
got RFO
forwarding it to connected nodes.. ok
status.. matching
matching..
answer sent to 192.168.0.1
status.. available
```

Figure 4.5: Screenshot of node A2



```
[emir@a3]$ python cleaner.py
got RFO
forwarding it to connected nodes.. ok
status.. matching
matching..
answer sent to 192.168.0.1
status.. working
downloading.. 192.168.0.4
executing blahblah
uploading.. 192.168.0.4
status.. available
```

Figure 4.6: Screenshot of node A3

Chapter 5

Evaluation

In this chapter there is going to be a discussion of the system created for the purpose of this thesis. The discussion is going to concentrate on the strengths and the weaknesses of the system. Also several proposals on how the system can be improved are going to be presented. But first there is going to be a quick look at the state of the distributed computing in the world today.

Today distributed computing is in its infancy. Yet the companies and the institutions around the world believe that it will change the world of computing. System for distributed computing should be:

- Secure
- Easy to use without constraining its users
- Offer consistent interface to different types of resources.
- Powerful and available for general use

Currently there is no system that fulfils all of the goals mentioned above but there are several projects which aim to fulfil those goals. One very interesting project is BOINC, which is developed at the USCA Berkeley. Its goal is to provide core client which would in turn download and execute project-specific application programmes. BOINC takes a bottom-up approach to fulfilling above mentioned goals. Other projects, most notably various Grid projects that use Globus toolkit seem to be taking top-down approach.

Now let's take a closer look at the system created for the purpose of this thesis. Unlike all of the systems for distributed computing that exist on the market today system created for the purpose of this thesis does not contain any central points. System is fully distributed. How many of the above mentioned goals does this system fulfil and what are the main weaknesses of the system?

The first point in the list above states that the system should be secure. No one is going to use a non-secure system. Security model for the system is presented in the chapter 3.3 but is as mentioned in chapter 1.5 not implemented.

The second point states that system should be easy to use without constraining the user. Currently the system of this thesis fails this point miserably. It is not at all easy to use. Its command line interface (CLI) is not intuitive and does not offer any help to the user. A Graphic user interface (GUI) needs to be implemented.

The system fulfils the third point which states that the users should be offered consistent interface to different types of resources. As it is now all the jobs are created in the same way - through the job description files.

The fourth point is probably the most important goal of all the systems for distributed computing. This point is only partly fulfilled. At the first glance the system is available for general use - user can use system for different types of jobs. Yet if the brokering in the system is examined more closely several deficiencies are going to be found. Firstly, the system does not support queuing. No support for queuing means that a node, which is already busy processing a job is not going to accept any new jobs for later execution. Secondly, rating system is only available for machines. Currently jobs can rate machines with the point system but machines can not rate jobs. Once a node on the network receives a job request it involves the matching algorithm. If the match is successful, machine sends answer to the job originator and waits. If a new job requests arrive during this "wait" time, the node is only going to forward this request to its neighbours. It would be desirable that the node would involve the matching protocols while being in the "wait" state. If the new job receives a higher rating than the old job, than the node is going to choose to process that job instead of the old one.

Solving of the above mentioned deficiencies would enable further improvement in the handling of the jobs at the nodes. For example a node is executing a job X with a priority 10. A job Y with a priority 15 arrives into node queue. Node notices that the job with higher priority than the current executing job is available in the queue. Job X is paused and job Y is started. When job Y has finished node un-pauses job X and starts executing it again. Third and final deficiency in the brokering algorithm has to do with its assumption that all the nodes contain only one CPU unit. Currently there is no way to define jobs that use multiprocessor machines.

All of these deficiencies need to be solved before this system can be considered for any serious use. Solving of these problems can probably be the subject of a new master thesis that could build upon this one.

Chapter 6

Discussion and Future Work

In the beginning of the nineties it was a multimedia computer. Later on it was Internet, 3G and Napster. Is the current buzzword of computer scientists and ordinary people - Grid and distributed computing - just going to be a fade or is it going to revolutionise supercomputing forever like its proponents suggest?

Distributed computing brings huge benefits. Cost savings and easier access to resources are some of these. But before these benefits are made clear to the ordinary users there are obstacles that need to be passed. When analyzing the situation in distributed computing there are two questions that need to be answered to be able to understand the current situation in the field of distributed computing. Firstly, what do users want? And secondly, what are they offered? Subtracting the second question from the first produces a list of the things that need to be addressed before distributed computing takes a step from being a buzzword to being one of the computing worlds many concepts. One concept that is not just cool (mentioning distributed computing right now usually produces "that's so cool" answer) but one that works, one that ordinary people and scientist take for granted. This will off course lead to distributed computing loosing its "cool" aura, things usually do that when users start taking them for granted, but computer scientists can probably live with that.

One thing that can be learned from Microsoft Windows climb to the monopoly in the operating system market is that the users want easy to use and consistent interface. There is no reason to believe that the distributed computing users would want something else. Another thing that can be derived from the operating systems wars (and even from the web browser wars) is that users want single system that handles all their needs. It appears that the majority of the users do not want specialized programs that do one thing but want more general systems. Finally, from the current trend of more and more people switching to GNU/Linux operating system it can be concluded that the users do not want dumbed down systems. The system should offer help to the users but it should not prevent these same users from using it in some more creative way than the system was originally intended for. Another thing that the users want, but usually do not realise, is security. No one is going to use a system if they are going to risk loosing their data. Either to a hard drive crash or to a competing company. Thus system for distributed computing should be:

- easy to use without constraining a user
- offer consistent interface
- be available for general use
- powerful
- secure

Let's look at what is available today. There is a plethora of small systems geared for solving one or two problems. They are easy to use and usually offer consistent interface. The problem is that they are not general, code is usually proprietary and they are not very powerful. Simply these small systems do not offer to the user what the user wants. One such system, distributed.net, was described earlier on in this report. However there seems to be some progress in this area. One very interesting project is previously mentioned BOINC which is developed at Berkeley. Its goal is to provide core client which would in turn download and execute project-specific application programmes.

While the systems mentioned above seem to be taking bottom-up approach, there is another group of systems that seem to be taking top-down approach. These systems are usually called Grids. Goal is to offer all above mentioned things that the user wants. The main problem with different Grid projects seems to be a lack of delivery. Lots of great ideas are crafted but very few are delivered in practice. Yet even in this area there seems to be lots of movement towards the goals mentioned above.

When it comes to the system developed for the purpose of this thesis there is a lots of room for improvements. Currently there is security model for the system, but it is not implemented in the code. Brokering layer contains also few deficiencies. Firstly, it does not support queuing. Secondly, it assumes that all machines on the network are one-processor machines. The brokering layer does not understand difference between serial and parallel jobs. Solving all these problems could probably be subject of a new master thesis that could build upon this one.

Today distributed computing is in its infancy. Yet it has to be concluded that distributed computing is not science-fiction anymore but reality today. It is extensively used around the world. From Scandinavia to Japan and the United States. Yet despite extensive use of distributed computing around the world, it seems that distributed computing is not yet ready for the primetime. It is still too hard to use for ordinary researchers who are not interested in mechanisms behind the systems. But in two or three years most of the problems are going to be solved and distributed computing is going to take a step from laboratory environments into production environments. This will be one much anticipated step as researchers today yearn for more computing capacity.

Appendix A

XML Schema

This appendix contains a XML Schema that define the language that is used in the implementation described in this document. These specifications are used both for validating and documenting.

```
<!-- 0.1 -->
<?xml version="1.0">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- Part 1                                                  -->
  <!-- This part of the document describes requirements part -->
  <!-- of the specification                                    -->

  <!-- This element describes operator -->
  <xs:simpleType name="operator">
    <xs:restriction base="xs:Name">
      <xs:enumeration value="lt"/>
      <xs:enumeration value="eqlt"/>
      <xs:enumeration value="eq"/>
      <xs:enumeration value="noteq"/>
      <xs:enumeration value="eqgt"/>
      <xs:enumeration value="gt"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- This element describes value as nonPositive integer -->
  <!-- It is overrided by "local" value when value should  -->
  <!-- be treated as something else                        -->
```

```
<xs:element name="value">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:nonNegativeInteger">
        <xs:attribute ref="unit"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<!-- Unit is attribute to value. This is definition of -->
<!-- of enum field unit                                 -->
<xs:attribute name="unit">
  <xs:restriction base="xs:Name">
    <xs:enumeration value="kHz"/>
    <xs:enumeration value="MHz"/>
    <xs:enumeration value="GHz"/>
    <xs:enumeration value="Mb"/>
    <xs:enumeration value="Gb"/>
    <xs:enumeration value="Tb"/>
    <xs:enumeration value="kbit"/>
    <xs:enumeration value="Mbit"/>
    <xs:enumeration value="Gbit"/>
    <xs:enumeration value="min"/>
    <xs:enumeration value="hour"/>
  </xs:restriction>
</xs:attribute>

<!-- This element describes point -->
<xs:element name="point>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:positiveInteger">
        <xs:attribute ref="per"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<!-- Per is attribute to value -->
```

```
<xs:element name="per" type="xs:positiveInteger">

<!-- This element describes complexType walltime -->
<!-- Walltime                                    -->
<xs:element name="walltime">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="value" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="point" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType cpu -->
<!-- CPU Clockspeed                         -->
<xs:element name="cpu">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="value" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="point" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType ram -->
<!-- Random Access Memory                   -->
<xs:element name="ram">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="value" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="point" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType hd -->
<!-- Available space on HD                 -->
```

```
<xs:element name="hd">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="value" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="point" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes comlexType net -->
<!-- Network connection speed             -->
<xs:element name="net">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="value" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="point" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType arch  -->
<!-- It overrides global types value/operator -->
<!-- with "local" value/operator              -->
<!-- Architecture                             -->
<xs:element name="arch">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:simpleType name="operator" minOccurs="1" maxOccurs="1">
        <xs:restriction base="xs:Name">
          <xs:enumeration value="neq"/>
          <xs:enumeration value="eq"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<!-- This element describes complexType os    -->
<!-- It overrides global types value/operator -->
<!-- with "local" value/operator              -->
<!-- Operating System                         -->
<xs:element name="os">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:simpleType name="operator" minOccurs="1" maxOccurs="1">
        <xs:restriction base="xs:Name">
          <xs:enumeration value="neq"/>
          <xs:enumeration value="eq"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType distribution  -->
<!-- It overrides global types value/operator         -->
<!-- with "local" value/operator                      -->
<!-- Linux distribution                               -->
<xs:element name="distribution">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:simpleType name="operator" minOccurs="1" maxOccurs="1">
        <xs:restriction base="xs:Name">
          <xs:enumeration value="neq"/>
          <xs:enumeration value="eq"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType release  -->
<!-- It overrides global type value with local   -->
<!-- value                                       -->
<!-- Architecture                                -->
```

```
<xs:element name="release">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="value" type="xs:decimal" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="operator" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes complexType requirements -->
<xs:element name="requirements" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="arch" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="os" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="distribution" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="release" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="cpu" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="ram" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="hd" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="net" minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="walltime" minOccurs="0" maxOccurs="unbounded">
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Part 2                                                  -->
<!-- This part of the document describes enviroment part   -->
<!-- of the specification                                    -->

<!-- This element describes path -->
<xs:element name="path" type="xs:string"/>

<!-- This element describes download  -->
<!-- Download file from this location -->
<xs:element name="download">
  <xs:complexType>
    <xs:sequence>
       <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
       <xs:element ref="path" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes execute -->
<!-- Execute this file              -->
<xs:element name="execute">
  <xs:complexType>
    <xs:sequence>
       <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="1"/>
       <xs:element ref="path" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes upload -->
<!-- Upload file to this location  -->
<xs:element name="upload" type="xs:string">

<!-- This element describes complexType enviroment -->
<xs:element name="enviroment" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="download" minOccurs="1" maxOccurs="unbounded">
      <xs:element ref="execute" minOccurs="1" maxOccurs="1">
      <xs:element ref="upload" minOccurs="1" maxOccurs="1">
    </xs:sequence>
  </xs:complexType>
</xs:element>



<!-- Part 3                                         -->
<!-- This part of the document describes owner part -->
<!-- of the specification                           -->

<!-- This element describes name                  -->
<!-- It is part of all other tags in this section -->
<xs:element name="name" type="xs:string">
```

```
<!-- This element describes signature -->
<!-- It is part of all other tags in this section -->
<xs:element name="signature" type="xs:string">

<!-- This element describes owner -->
<!-- Job owner                   -->
<xs:element name="owner">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1">
      <xs:element ref="signature" minOccurs="1" maxOccurs="1">
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- This element describes other -->
<!-- Those who validate          -->
<xs:element name="other">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name" minOccurs="1" maxOccurs="1">
      <xs:element ref="signature" minOccurs="1" maxOccurs="1">
    </xs:sequence>
  <xs:complexType>
</xs:element>

<!-- This element describes complexType owner -->
<xs:element name="identification" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="owner" minOccurs="1" maxOccurs="1">
      <xs:element ref="other" minOccurs="0" maxOccurs="unbounded">
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Appendix B

Matching Tables

The matching algorithm divides variables that are to be matched into two groups. These two groups are numbers and strings. Numeric variables can be matched with the help of six different operators while the string variables are matched with only two different operators (equal and not equal). Tables B.1, B.2 and B.3 present matching decisions that the matching algorithm makes when it deals with variables consisting of numbers. As can be seen the matching decisions are influenced by the operators and the values of the variables involved. Table B.4 represents matching decisions for string variables.

machine/job	$\geq$	$>$
$\geq$	Always	Always
$>$	Always	Always
$<$	If $m.Value > j.Value$	If $m.Value > j.Value$
$\leq$	If $m.Value \geq j.Value$	If $m.Value > j.Value$
$=$	If $m.Value \geq j.Value$	If $m.Value > j.Value$
$\neq$	If $m.Value < j.Value$	If $m.Value \leq j.Value$

Table B.1: Matching decisions for number values (part 1)

machine/job	$<$	$\leq$
$\geq$	If $m.Value > j.Value$	If $m.Value \geq j.Value$
$>$	If $m.Value > j.Value$	If $m.Value > j.Value$
$<$	Always	Always
$\leq$	Always	Always
$=$	If $m.Value < job.Value$	If $m.Value \leq j.Value$
$\neq$	If $m.Value \geq j.Value$	If $m.Value > j.Value$

Table B.2: Matching decisions for number values (part 2)

machine/job	$=$	$\neq$
$\geq$	If $m.Value \leq j.Value$	If $m.Value > j.Value$
$>$	If $m.Value < j.Value$	If $m.Value \geq j.Value$
$<$	If $machine.Value > j.Value$	If $m.Value \leq j.Value$
$\leq$	If $m.Value \geq j.Value$	If $m.Value < j.Value$
$=$	If $m.Value = j.Value$	Never
$\neq$	Never	If $m.Value \neq j.Value$

Table B.3: Matching decisions for number values (part 3)

machine/job	$=$	$\neq$
$=$	If $m.Value = j.Value$	If $m.Value \neq j.Value$
$\neq$	If $m.Value \neq j.Value$	If $m.Value = j.Value$

Table B.4: Matching decisions for string values

List of Figures

List of Tables

Bibliography

[1] *Matchmaking: Distributed Resource Management for High Throughput Computing.* Proceedings of the 7th IEE International Symposium on High Performance Computing, July 1998.

[2] M.R. Horton & R. Adams. *RFC1036: Standard for interchange of USENET messages*, December 1987.

[3] J.C.R. Licklider W. Clark. On-line man computer communication. ., August 1962.

[4] drscholl@users.sourceforge.net. *OpenNap protocol specification*, April 2000.

[5] drscholl@users.sourceforge.net & cyberalien@users.sourceforge.net. *SlavaNap protocol specification*, March 2002.

[6] A. Oram et al. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology.* O'Reilly & Associates, March 2001.

[7] Ian Foster. What is the grid? a three point checklist. *Grid Today*, July 2002.

[8] S. Garfinkel. *PGP: Pretty Good Privacy.* O'Reilly & Associates, March 1995.

[9] M.R. Horton. *RFC0976: UUCP mail interchange format standard*, February 1986.

[10] B. Kantor & P. Lapsley. *RFC0977: Network News Transfer Protocol*, February 1986.

[11] P. Albitz & Cricket Liu. *DNS and BIND.* O'Reilly & Associates, 4th edition, April 2001.

[12] T. Klingberg & R. Manfredi. *Draft Gnutella Protocol Specification 0.6*, June 2003.

[13] P.V Mockapatris. *RFC1034: Domain names - concepts and facilities*, November 1987.

[14] B. Nowicki. *RFC 1094: Network File System Protocol specification*, March 1989.

[15] Ann Onymous. Distributed policy management and comprehension with classified advertisments. *Sciences Technical Report*, (1481), 1997.

[16] E.T. Ray. *Learning XML.* O'Reilly & Associates, 2nd edition, September 2003.

[17] R. Shirey. *RFC2828: Internet Security Glossary*, May 2000.

[18] S. Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* Anchor, August 2000.

[19] W. Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, 3rd edition, August 2002.

[20] ITU Study Group VII. *Recommendation X.800: Security Architecture for Open Systems, Interconnection for CCITT Applications*, August 1991.