A Distributed Decentralised Information Storage and Retrieval System

Ian Clarke
Supervisor: Dr Chris Mellish

Division of Informatics
University of Edinburgh
1999

Abstract

This report describes an algorithm which if executed by a group of interconnected nodes will provide a robust key-indexed information storage and retrieval system with no element of central control or administration. It allows information to be made available to a large group of people in a similar manner to the "World Wide Web". Improvements over this existing system include:

- No central control or administration required

- Anonymous information publication and retrieval

- Dynamic duplication of popular information

- Transfer of information location depending upon demand

There is also potential for this system to be used in a modified form as an information publication system within a large organisation which may wish to utilise unused storage space which is distributed across the organisation.

The system's reliability is not guaranteed, nor is its efficiency, however the intention is that the efficiency and reliability will be sufficient to make the system useful, and demonstrate that such a system is feasible.

This report also describes several experiments designed to measure the efficiency and reliability of such a network. These are performed upon a simulation of a working network written in the Java programming language. Improvements over the existing "World Wide Web" are also highlighted in this report. The algorithm is considered a prototype, and areas for further study and potential improvement are highlighted throughout this report.

Contents

Chapter 1

Introduction

1.1 Freedom

> "I worry about my child and the Internet all the time, even though she's too young to have logged on yet. Here's what I worry about. I worry that 10 or 15 years from now, she will come to me and say 'Daddy, where were you when they took freedom of the press away from the Internet?" -Mike Godwin

At present the Internet is seen as a hot-bed of anarchy, beyond the controls of individual governments, a system which ensures free-speech and free-thought by its very nature. Unfortunately, while this seems true at present, it is caused more by a lack of knowledge about the technology involved, rather than being a feature of the technology. It is actually the case that the Internet could lead to our lives being monitored, and our thinking manipulated and corrupted to a degree that would go beyond the wildest imaginings of Orwell. As an example, most web browsing software at present maintains a list of all web-sites so-far visited. That information could easily be sent to a central location on a daily basis. It would then be possible to build up a detailed picture of an individual without their consent or knowledge. Furthermore, it is much easier to monitor vast amounts of email than it is to monitor telephone or postal communications. The mention of specific words of phrases could flag an email for further investigation. This will become yet easier as natural language interpretation systems grow more sophisticated.

It is not just the information consumers that are at risk. Every piece of information on the Internet can be linked to the owner of the server that hosts the information. There is an indelible electronic trail from the information to those that are responsible for it. In many situations this trail is a restriction upon free-speech, as fear of retribution can (and often does) serve as an effective censor.

1.2 A distributed decentralised Internet

It is curious that most of the creations of man follow a design philosophy which is alien to that employed in most biological organisms, including man himself. Whereas evolution has discovered that successful systems tend to be decentralised with only the minimum centralised control possible, the systems created by man tend to have a highly centralised architecture. Computers have central processing units, computer networks have hubs, and the operation of a car depends upon that of a small fuel pump. While it could be argued that a person is centrally controlled by the brain, if we look at each of the sub-systems within a person, we discover a degree of redundancy. Large portions of the brain can be removed with little significant degradation in performance. Compare this with one of man's most complex creations, the computer. Remove or damage even a microscopic component of a computer, and in the vast majority of cases the computer will stop working completely. There are examples where some form of redundancy has been incorporated into a design, however in comparison to nature's example these are clumsy and inelegant. Many people point to the Internet as an example of a distributed, decentralised, and robust architecture. This is not the case. Internet Protocol (IP) addresses are passed down from a centrally controlled system. The Domain Name System (DNS) upon which the modern Internet depends is ultimately controlled by one organisation. In most countries there is a small number of backbones upon which that country depends to provide access to the Internet. There are frequent failures in this system (at the time of writing there has already been at least one significant failure in the D.N.S in 1999 alone[1]), and as the Internet becomes more important these failures will become more significant. Perhaps it is possible to design a system which is decentralised, and distributed, while retaining the elegance of natural solutions.

[1] On the 12th March 1999 several of the servers which store details of domain name ownership became corrupted and began reporting that domains such as "microsoft.com" and "yahoo.com" were available for registration. At the time of writing the cause is not yet known.

Chapter 2

Aims

A key-indexed storage and retrieval system such as the World Wide Web, or the system described by this report, allow anyone connected to the Internet to retrieve information corresponding to a key. The methods and limitations upon how information can be added to the system, and limitations upon how the information is accessed are specific to individual system. The aim of this project is to devise a key-indexed information storage and retrieval system with the following properties:

1. The system should have no element of centralised control or administration
2. It should provide anonymity to both providers and consumers of information
3. The system should be robust in handling hardware and software failure
4. It should adapt to changing demands and resources in an efficient manner
5. Its performance should be comparable to that of existing mass information distribution systems such as the "World Wide Web"

And also to evaluate the performance of this system by means of experimentation.

Chapter 3

Related Work

In this section I examine several examples of distributed information storage and retrieval systems, and use these to justify the aims of this project.

3.1 The Domain Name System (DNS)

3.1.1 Description

While most computers are referred to by their relatively user friendly "domain" names (such as www.dcs.ed.ac.uk), the underlying routing mechanisms in the Internet require the Internet Protocol address (IP address) of a computer before being able to transfer messages. An IP address consists of 4 bytes, generally represented as 4 numbers separated by full stops. The Domain Name System (or DNS) is a world-wide database used to convert the names of computers on the Internet (such as www.dcs.ed.ac.uk) into IP addresses (such as 192.168.134.100). Rather than retain all of this information in a central database, different organisations have been given responsibility for parts of the database, and some have delegated sub-parts of that database further. For example, in the United Kingdom "Nominet" has responsibility for all domain names ending with ".uk". It has sub-delegated all domain names ending with ".ac.uk" to the Joint Academic Network (JANet). In turn, JANet has delegated all domains ending with ".ed.ac.uk" to the University of Edinburgh, and so on. Each organisation must either handle administration of the database for their sub-domain (such as additions, amendments, and deletions), or sub-delegate it to another organisation.

Given a domain name, the client obtains the corresponding IP address by first querying one of several "root" nodes, these will respond with the location of the appropriate DNS server. The query is then sent to that server which refers the client to further DNS servers lower in the hierarchy until eventually the computer is found (see figure 3.1). The efficiency of this system is improved by means of caching. Rather than querying the root DNS servers on every occasion, queries will first be directed to local DNS servers which maintain a

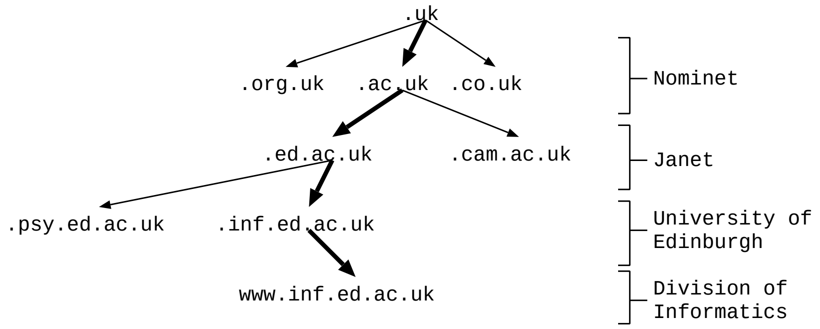


Figure 3.1: Searching for information in the Domain Name System

cache of recently accessed domains. If a domain is requested that is in the cache then the client can be referred directly to the relevant DNS server, improving the speed of the query, and reducing the load on DNS servers higher in the hierarchy.

3.1.2 Discussion

While the Domain Name System is distributed, it is still centrally controlled. There is little, from a technical perspective, to prevent the owners of the root domain name servers from reclaiming the ability to administer sub-domains of the system, or from accidentally stopping the entire system from operating[1]. Further, any of the sub-administrators who have been given control over parts of the database have the same power over the sub-domains that they administer. This could include charging unreasonable prices for additions or changes to the database, or imposing some form of censorship over the system (neither the anonymity of providers or consumers of information is protected in any way). In many cases the methods used to allocate domain names to organisations and individuals is causing serious problems. Most organisations charged with administering the Domain Name System allocate domains on a first-come-first-serve basis. This has lead to speculative registration of large numbers of domains by companies hoping to sell on the domain names at a greatly inflated price. The Internet is a public utility, formed from the collective efforts of many people and organisations. Thus to require a small number of organisations to have such control over the system, and to make it possible for them to profit greatly through exploitation of their monopoly, reduces the overall reliability of the Internet as a means to freely and inexpensively exchange information.

3.2 The World Wide Web

3.2.1 Description

The World Wide Web is the most visible example of a computer based key-indexed information distribution system. It makes use of the HyperText Trans-

[1](see footnote 1, page 5).

fer Protocol (HTTP), which in turn sits above TCP/IP[2]. The key used to access a piece of information is known as a Uniform Resource Locator (URL), and as the name suggests, it essentially represents the computer on which the information resides, as well as the location within that computer where the information can be found. The format of a URL is as follows:

```
http://www.dai.ed.ac.uk/ed/aisoc/index.html
\__A__/\______B________/\_______C_________/
```

A - The protocol used (normally HTTP however FTP and FILE may also be specified)
B - The computer on which the resource is located
C - The location of the file on the computer (a directory path)

The Domain Name System is first used to obtain an IP address for the computer (see section 3.1). A TCP connection is then opened to that computer and the full URL is sent to that machine as part of a "GET" request. That machine will then either return an error message indicating that the requested information is not available, or it will return the information itself. Generally information on the world wide web is stored in HyperText Markup Language (HTML), however this is by convention only. HTTP expects a reliable stream based communication medium which is provided by TCP.

If someone wishes to make information available via the world wide web they must place that information on a computer with a permanent Internet connection, that is running HTTP server software. The URL required to access that information will be determined by the domain name of the computer the information is on, and also by the location within the file-system on that computer. In order to find the URL of a piece of information given the subject of that information, there are several unreliable methods which may be employed. Firstly it could be assumed that the name of the computer on which the information is stored is indicative of the subject of the information (as is sometimes the case). For example, to find information on the Linux operating system, we could look at the URL http://www.Linux.org/. Unfortunately due to the hap-hazard first-come-first-serve method most domain names are allocated, this method is far from reliable. Secondly, a "search-engine" such as Yahoo (http://www.yahoo.com/) which maintains a central index of webpages may be used to retrieve information. This is the most common method to locate information on the Internet where a URL is not already known, however the organisations which operate these services have an opportiunity to censor information, and often websites which have paid a fee are given preferential treatment. Further, the small number of these search engines create a single point of failure that decrease the reliability of the WWW if they are depended upon exclusively.

[2] TCP stands for Transport Control Protocol, IP stand for Internet Protocol. IP provides an unreliable packet-based communication medium, TCP sits above this providing a reliable stream-based communication medium.

3.2.2 Discussion

Normally each piece of information on the World Wide Web is stored once on an individual computer on the Internet. Every time anybody wishes to view that information a request is sent to that computer and it returns the relevant piece of information. This model works well when there are relatively few requests, however if a piece of information becomes increasingly popular then the computer possessing the piece of information can become swamped by requests, preventing many from obtaining the information. One solution is to move the information to a more powerful connection to the Internet, however this often necessitates a change in URL. Also, frequently the increase in demand for the information may be short-lived[3]. There are several solutions to this problem.

The first is mirroring. This is a measure that can be taken by the provider of the information to reduce the load on their HTTP server. If it is anticipated that there may be a high volume of requests for a particular piece of information, then several other computers on the Internet can offer to retain copies of the information. People may then obtain the information from those systems instead, reducing the load on the main server. This method is widely used, however mirrors must be set up specially, and require some organisation. If high load is imposed on a system without warning then there may be a long delay before suitable mirrors can be set up. Also, to an extent, this system relies on the good-will of the user to make use of the mirrors rather than the main site.

The second method is caching. This is a measure that can be taken by the information consumer. It involves keeping a local cache of recently accessed documents, so that if these are re-accessed before they expire from the cache then the HTTP server on which the information resides need not be queried. There are several levels at which caching may be implemented. Most modern web-browsing software maintains a local, per-person cache. Further, many organisations now maintain an organisation-wide cache of web pages. Caching does result in an increase in web performance, and a decrease in traffic, however it requires significant effort on the part of the information consumer, or the organisation through which they access the Internet. Also, there is the matter of choosing the appropriate level at which to cache the information. Generally this is at the level of the web-browser and the organisation, however it may be desirable to do this on a wider scale. Further, with this caching mechanism, there is the possibility that an unscrupulous organisation could supply users with modified information. There is little to prevent a time-saving web-cache from becoming an electronic censor of information, and indeed this has happened in countries such as China where the government openly censors information on a wide scale.

The World Wide Web offers little anonymity to either producers or consumers of information. If it is possible to retrieve a piece of information then an IP address for the server on which the information is stored must have been found.

[3]This has become known as the "SlashDot" effect. http://slashdot.org/ is a popular news site on the world wide web. It was discovered that when a hyperlink was placed on the SlashDot site there would be a dramatic increase in demand for that piece of information. If the computer that the information was stored on was not sufficiently powerful, the web server would fail preventing access to the information.

If this is the case then that IP address may be matched to an organisation or person. Even if that person did not create the information, they are responsible for its availability, and thus could be forced to remove the information. There are cases where intellectual property law has been used to force information to be removed from the World Wide Web[4].. Similarly, when someone connects to a remote server to retrieve a web page, their IP address becomes known to the remote server. This can be used with other information to determine what country, even what organisation the user is accessing the WWW from. Further, by correlating this information (possibly in conjunction with other organisations) a vast amount of information could be collected about a person, determining which web sites they visit most frequently, and thus their political views and interests, even their sexuality, could be inferred. As usage of the World Wide Web increases as people rely upon it more and more as a source of news and information, concerns such as those raised here will become more important.

3.3 UseNet

3.3.1 Description

UseNet is one of the older systems on the Internet. It is a world-wide "bulletin-board" discussion system where people may post articles, and respond to the postings of others. The system is divided up into many "newsgroups" each of which contain discussions relating to a specific topic called "newsgroups", almost every conceivable subject has an associated newsgroup on Usenet. The system is distributed across many Usenet servers worldwide. Most allow users to post new news-items to newsgroups, and to read existing postings. At regular intervals the newsgroups will distribute the ID numbers of new postings to their neighbours, who will distribute them to their neighbours in the same manner, until they have propagated over all UseNet servers (see figure 3.2). In this way, each Usenet server contains a reasonably up-to-date copy of all messages posted. Most UseNet servers are configured to delete old messages, although the creator of a posting can also specify an earlier date for deletion.

3.3.2 Discussion

While Usenet succeeds in being a distributed, and largely decentralised system, it is extremely inefficient. The requirement that every message posted to the system must be distributed to each and every server on the system means that each server must be capable of storing huge quantities of data (if each article is not to expire before anyone has a chance to read it). Furthermore, the bandwidth required to transfer the new messages between servers is extremely high. There are also many security problems with Usenet. For example, if you are the

[4]An example is when the Church of Scientology forced the closure of a web site which criticised them on the basis that it contained information to which they owned the copyright. Even when an organisation is unlikely to win a court case against an information provider, if they have sufficient resources they can keep the case going until their opponent runs out of funds.

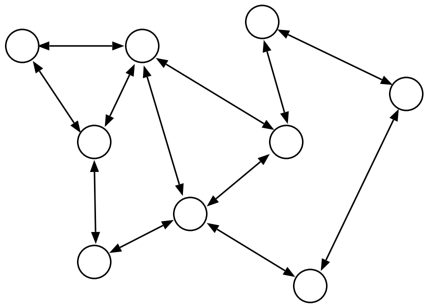


Figure 3.2: Propagation of postings between UseNet servers

author of a message, you can send a “cancel” message to a Usenet server which will cause your message to be removed. Unfortunately it is extremely easy to send a cancel request for someone else’s article, pretending to be them. There have been instances where this has been used to delete messages on political grounds. Usenet is a legacy system. Were it to be completely re-implemented today it would be done using the HTTP protocol. Already there are many discussion systems which operate over the WWW in a much more efficient, and much more flexible manner. It does serve as an example of one method to achieve a distributed decentralised information distribution system, but does so at the price of extreme inefficiency, and lack of security. Having said that, of all the systems described in this section, the operation of Usenet comes closest to that of the system described in this report. See [1] for further details of the operation of Usenet.

3.4 The Eternity Service

3.4.1 Description

The Eternity Service is a system designed with the intention of storing data over long periods of time, and being resistant to forms of attack such as administrator bribery and legal challenge. The following is an extract from [3, node9] describing the operation of this system:

> Each hardware security server will control a number of file servers. When a file is first loaded on to the system, it will be passed to the local security server which will share it with a number of security servers in other jurisdictions. These will each send an encrypted copy to a file server in yet another jurisdiction.
>
> When a client requests a file that is not in the local cache, the request will go to the local security server which will contact remote ones chosen at random until one with a copy under its control is located. This copy will then be decrypted, encrypted under the requesters public key and shipped to him.

3.4.2 Discussion

While the system described above does achieve the aim of securing information, and making it extremely difficult to remove the information from the system, or even finding the location of the information in the first place, it does not address efficiency issues. If there are a large number of servers employed in the system (as there undoubtedly would be if it were a system in world-wide use) then the mechanism described for randomly polling servers until the information is found becomes highly inefficient. In addition, the mechanism for expanding the network of servers involved in the system, while maintaining the strict levels of security, are not specified, and may prove problematic.

Chapter 4

Initial Ideas - A redundant hierarchy

4.1 Description

Initially, the model of a binary search tree was examined as a potential model for the solution (see [2, chapter 13] for a description of binary search trees). These provide an efficient means to store and retrieve information which is referenced by a key (such as an integer). Basic operations such as storing and retrieving information require $\Theta(\ln n)$ time to complete where n is the number of items in the tree. Thus binary trees can handle large amounts of data in an efficient manner.

It would be relatively simple to have a group of interconnected computers behave as a binary tree, forwarding requests on to the appropriate nodes until the information is found, and then returning the results to the user that requested it. Unfortunately such a system would be extremely susceptible to failure. If a node high up in the tree were to fail, then all data stored on nodes lower down would be inaccessible. Further, the owner of the top-most node in the tree would essentially have control over the entire system as all requests for information would initially go to it. Because of this problem a method was considered whereby rather than each node in the hierarchy being an individual computer, each node would be a group of computers which would share the responsibilities of that node (see figure 4.1). Nodes in the network which had more responsibility would consist of more computers, so that the responsibility of each individual computer was limited.

4.2 Discussion

The redundant hierarchy solution to the problem raised many questions about how the system would operate. Among them were:

- How would new computers be added to the system in a fair manner?

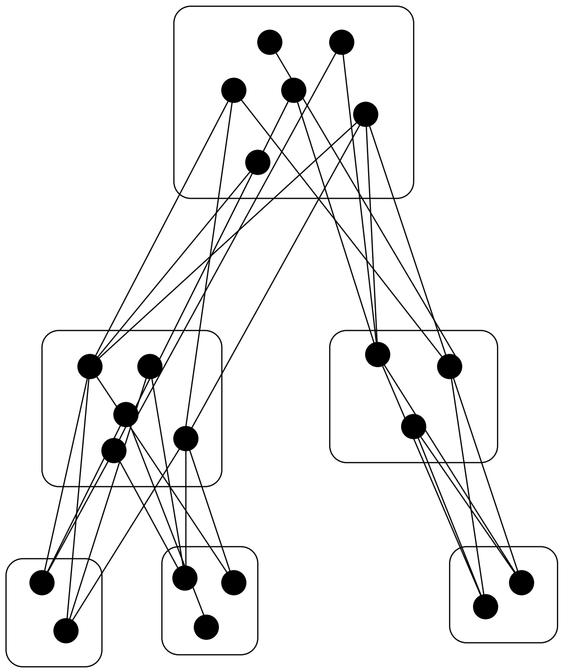


Figure 4.1: A “Redundant Hierarchy”

- How would the computers be organised within each node in a distributed manner?
- How would the system adapt to changing demands on information?
- How would the system deal with the merging of two networks into one?

It was decided that rather than attempting to answer these questions, the abundance of issues raised suggested that an alternative approach was required.

Chapter 5

An adaptive network

5.1 General Description

Navigation in prehistoric society was used as inspiration for the location of information in an adaptive network. In such a society, there is no central government, and no maps, yet is was possible for people to navigate long distances even without these guidelines by relying on the advice of those they encountered. It was reasonable to assume that while people would have detailed information on how to reach locations close to them, say, they would have less detailed information about locations further away, and only vague details of distant locations such as the general direction to travel and a very approximate distance. By starting at a location, asking advice about where another place is, and following that advice, you are likely to find yourself somewhere closer to your destination. You can then obtain more detailed advice, and eventually you will arrive at your destination.

It is this principle that is exploited in an adaptive network, however rather than finding a location by getting closer and closer to it geographically, we get closer and closer to a piece of information[1]. In an adaptive network each computer or node that participates in the network stores some information locally, but also contains information about other nodes in the network. If a node receives a request for a piece of information that it does not have locally, it will forward the request to whichever node is most likely to have the information according to its information. When the information is found, it is passed back through the nodes that originally forward on the message, and each is updated with the knowledge of where the information was stored. In addition, the information that has been retrieved will also be cached on the node locally for a short time so that future requests for it may be answered immediately without forwarding on the request. Nodes are more likely to receive requests for information that is similar to the information they possess - just as people who live near a given location are more likely to be asked directions to it than people who live further away. Because of this, and the fact that nodes are updated with information

[1] For a more detailed discussion of what it means for two keys to be "close" see section 5.2.2.

about the location of information when they forward on requests, nodes that are (or contain information that is) close to other nodes are more likely to contain information about what they store. This means that a request for a piece of information will get closer and closer to the information sought, just as the prehistoric traveler will get closer to, and eventually reach their eventual destination.

The local caching of information on nodes also has a variety of effects. Firstly, if there is high demand for a piece of information among a certain group of nodes, then that information is likely to be cached on one or more nodes at or near that location. As this information is locally cached few of the requests will reach the original source of the information, and consequently this information may be removed from that node due to lack of requests (see section 5.2.1 for a discussion of information removal from the system). By this mechanism, information can move to a part of the network where it is in demand. So, for example, a piece of information, originally on a node in America, but popular in Japan, will eventually move to Japan reducing the need for cross-Pacific Internet traffic. The second effect of local caching is that a piece of information that suddenly becomes extremely popular will be duplicated over surrounding nodes. This duplication will continue until requests for that information are sufficiently dispersed that were there to be any further dispersion, the information would start being removed from nodes (in which case the concentration of requests would increase once more).

5.2 Considerations

5.2.1 Criteria for information removal

It is essential that there is some method by which information may be removed or "culled" from the network to prevent all available space on the network from being filled with potentially useless information. The criteria for information removal must be selected with caution to avoid the possibility that whatever mechanism is used can be employed as a form of censorship, it must also follow the principles of simplicity and decentralisation that we hope to embody in the system as a whole. As such, each node must make an individual decision as to what information should be removed.

The information removal criteria that has been decided upon is one of information popularity. If it becomes necessary to delete a piece of information on a node then the least-recently-accessed piece of information is deleted. This has the following advantages:

- This is a simple transparent procedure that can be implemented easily (see section 5.3.6)
- Information that nobody is interested in will be removed from the system, preventing seldom-accessed information from taking up space on the system in place of information of interest to a wider audience

- When combined with the "caching" effect of the adaptive network, it allows for information to move to parts of the Adaptive Network where it is accessed more frequently, eventually being deleted from its original location

The fact that the network judges the "worth" of information purely in terms of how many requests there are for it does raise some interesting questions. Using the World Wide Web as an example, one of the most popular types of information downloaded from the web is pornography. By the mechanism employed by this system, this material would be considered more valuable than, for example, an academic paper. While this may initially seem a disadvantage, it does mean that the systems priorities will reflect the interests of the users of the system as a whole.

It may seem here that information of minority interest risks being swamped by information of greater public interest, however this is not the case. Consider a situation where a small organisation (such as a university department) has placed information into the system which is primarily of interest to people within the department. Given that this information will not be requested very frequently relative to information of interest to the wider public, it might be imagined that it would be removed from the system, however this is not the case. While there are few requests for the information, they would all originate from a small number of nodes, and thus the information will be of greater importance to those nodes. The result is that the information is likely to be cached on nodes within the department where it will remain available to those who wish to view it. An issue does however remain, that information of interest only to a geographically dispersed minority may be dropped from the system. It should be remembered that the system is designed to serve the needs of the public as a whole, and as in any such system, the interests of very small minorities must take second place. It should also be noted that due to the speed at which storage capacity is increasing (by some estimates, it is more than doubling every year) the total information output of humanity is not growing at this rate in the long term, and thus it is possible that information removal from the system will be an infrequent event that effects only data that is virtually never accessed.

5.2.2 The nature of keys

The system requires a mechanism by which given a key, either the corresponding data, or the address of another node which may contain the data, can be found. To achieve this we must place some limitations on the nature of the keys we use. The first obvious property is that we can decide whether two keys are equal, in the case of strings, integers, and virtually any conceivable key that me might choose, this will be the case. The second property is that we must be able to establish a notion of "closeness" for keys. These two requirements are summarised by the functions:

```
Returns True if and only if a is closer to c than b:
  boolean ← compare(Key a, Key b, Key c)
```

```
Simple equality test, returns true if a is equal to b:
  boolean ← equals(Key a, Key b)
```

Unlike equality, the notion of closeness is not easily defined. To illustrate this, let us imagine that we have 4 keys, call them A, B, C, and D. We should be able to say "A is closer to B than D is". But what exactly do we mean by this 3-way relationship "is closer to"? There is a wide variety of ways in which the 'closeness' of two keys to each other can be decided. If we are dealing with, say, 64 bit integers, A, B, and C, where $Xval$ is the integer value of a key X, we could use the test $Aval - Cval < Bval - Cval$.

However, we could chose to split each key into 2 32 bit integers - so that each key has two integer fields, $xval$ and $yval$ - and treat each as a coordinate in Cartesian space. We would then use the test:

$$\sqrt{(Axval - Cxval)^2 + (Ayval - Cyval)^2} < \sqrt{(Bxval - Cxval)^2 + (Byval - Cyval)^2}$$

Of course, we need not stop here, we could treat each key as 3, 4, or more integers, and measure distance between them in the appropriate dimensional space. Yet we need not limit ourselves to keys in Cartesian space, we could treat them as polar coordinates, or something completely non-geometrical. Instead of using integer keys, we could use strings, or even binary trees. So what common thread runs through all of these to define what closeness is?

For our purposes we can define closeness in terms of the following two axioms[2]:

- "If A is closer to B than D is, and E is closer to B than A is, then E is closer to B than D is" and
- "If A equals B and D does not equal B, then A is closer to B than D is"

The first axiom is similar to the concept of "transitivity", however rather than it being a relation between two objects, it is a relation between three. The first axiom is nescessary so that given a list of keys, we can identify which is the closest to another "target" key without comparing every pair of keys (which, for a list of keys of length n would require n^2 time). Rather, we can pass through the list once maintaining a "best so far" value, so that if a key is compared to this "best" value, and found to be closer to our target key, then it can be assumed that it is closer to the target key than any node previously compared to it, it would then become the new "best" key for subsequent comparisons. This requires only n time to find the closest key. The second axiom is required to link the concept of closeness to the concept of equality (otherwise there would be no destinction between "closeness", and its opposite - "far-away-ness" as defined by the first axiom).

5.3 Architecture

An Adaptive Network has the following properties:

[2]The use of the term "axiom" in this context is drawn from the "Extended ML" language which, in addition to specifying the inputs and outputs to functions, allows the relationship between them to be expressed in mathematical terms.

- Each node in the network is identical in terms of authority and capability
- Each node need not use identical algorithms provided that they behave in the specified manner on receiving each type of message
- The system robustly handles incorrect or lost messages
- Information will be duplicated and moved around the network to optimise information retrieval

This design in split into two parts. Firstly there is the specification of the 4 different message types, what each means, and what behaviour will be expected of nodes on receiving each particular message. Secondly, there is the algorithms used to achieve this behaviour within each node. Variations in the algorithms used, and even the details of their operation, are possible, without changes to the message types. This means that different nodes in the network may have slightly different behaviours (possibly due to user preferences), but through conformance to a common, flexibly specified, message interface, they may all cooperate.

5.3.1 Message types

Any computer on the Internet may send a message to any other computer on the Internet[3] provided that they know the IP address of that computer. In an Adaptive Network, there are four types of message that can be sent between nodes in the network:

- Data Request
- Data Reply
- Request Failed
- Data Insert

Each message has an ID which should be unique to each operation (such as a data request, or a data insertion) and a "time to live" (T.T.L) which indicates how many times the message should be passed on. This value is decremented every time a node passes on a message. If a node receives a message with a T.T.L of 0 then it should generate a Request Failed message and return this to the sender of the message (obviously this message should not inherit the T.T.L of the message that initiated it!). In addition each message records who sent the message (but not who initiated the request), and who the recipient is. At this point the difference between the initiator of a request, and the sender of a message should be made clear. A node that initiates a request decides what the ID of the message and initial T.T.L should be. It then sends the request to another node, and thus they are the first sender, however if the recipient of

[3] An exception to this is if a computer is behind a "firewall", a security measure sometimes employed to protect a network from malicious tampering from elsewhere on the Internet. Firewalls must be configured to allow particular types of message through.

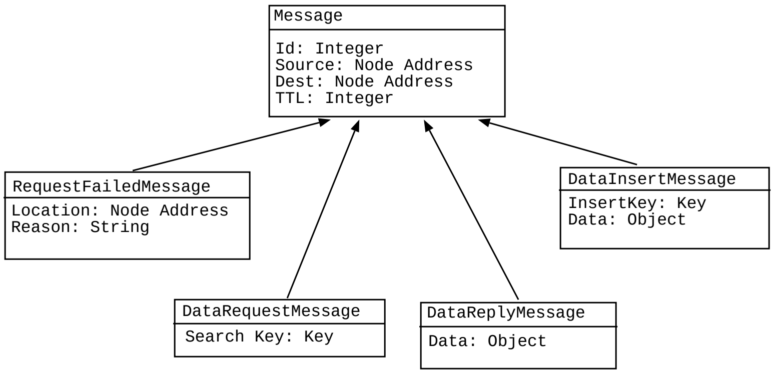


Figure 5.1: Field inheritance of different message types

that message forwards it to another node then they become the sender. The identity of the initiator of a request is not contained in a message. Even in the case of the first message there is no way to know that that computer that sent the request was not merely forwarding it from another source. This allows the initiator of a request to remain anonymous.

Figure 5.1 shows the various fields transmitted within each message type. As can be seen, the generic Message contains ID, TTL, source and destination fields, and these are inherited by all message types (but are not shown in the figure).

5.3.2 Data Request

The first message type is a request for the data corresponding to a particular key. If the node receiving the message has the requested data, then it should create a ***Data Reply*** message containing the desired data (this message should inherit the ID and T.T.L of the Data Request message). If the node does not have the data then it should forward the Data Request on to another node which is more likely to possess the data. A node should also store the ID of this data request and several other details for the purposes of handling received Data Reply and Request Failed messages. If a Data Request is received with an ID that has been seen before, then a "backtracking"[4] Request Failed message should be returned to the sender of the message.

5.3.3 Data Reply

A Data Reply message is a response to a Data Request which returns the requested data to the node that initiated the request. A node should only receive a Data Reply corresponding to a Data Request that they have forwarded previously, any other Data Reply messages are ignored. The Data Reply should be forwarded to the node that the original Data Request was received from (thus the Data Reply will eventually return to the node that initiated the data

[4] See section 5.3.4.

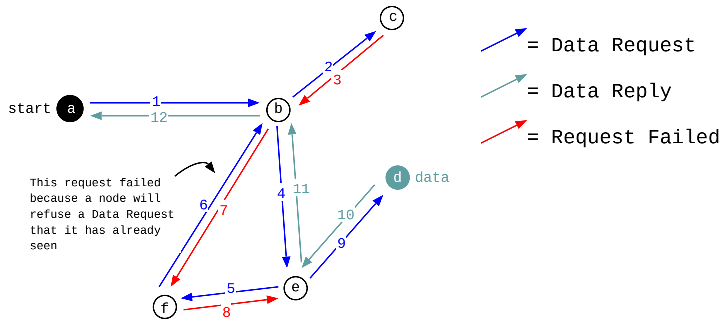


Figure 5.2: Messages sent once node **a** receives a request for information stored on node **d**

request). The node should retain a local copy of the data which has been forwarded so that future requests for that data may be answered immediately. Additionally, the node should periodically store a reference to the node that the DataReply is forwarded to.

5.3.4 Request Failed

A Request Failed message should only be received by a node that has forwarded on a corresponding Data Request message. Broadly there are two types of Request Failed messages, those that should "backtrack" and those that don't. A Request Failed message generated as a result of a message timing out it should simply be forwarded to the node that sent the original Data Request to this node. If a backtracking Request Failed message is received then a Data Request message should be sent to the "next best" node (this message should inherit the ID and T.T.L of the Request Failed message). If all nodes have been explored in this way then a Request Failed message should be sent back to the node that sent the Data Request. Generally, all Request Failed messages other than a time-out will be backtracking requests. A request failed message will also contain a description of the reason for the failure.

5.3.5 Data Insert

A Data Insert message is used to add some data to the network. On receiving a Data Insert message, a node should store the key and data contained in the message, and forward the message on to whichever other node is most likely to have similar keys to the one in the Data Insert message. Data Insert messages will not travel indefinitely through the network as their time to live counters will eventually reach 0, upon which a Request Failed[5] message indicating a timeout will be returned to the initiating node. The initial T.T.L of a Data Insert message should be low, if a node is asked to pass on a Data Insert message with a high T.T.L it can simply reduce the T.T.L to a more reasonable level

[5]Of course, in this case the Request Failed message does not actually indicate a failure, it is inevitable that a Data Insert message will result in a timeout.

before forwarding the message. This allows the prevention of exploitation of the system. If a node receives a Data Insert message that it has seen before, it should send a non-backtracking Request Failed message to the sender, this prevents Data Insert messages getting into pointless loops[6].

5.3.6 Node Architecture

In the descriptions given in section 5.3.1 the tasks that should be carried out by each node on receiving a particular message were described in general terms, in this section these tasks are described in more detail, along with how they may be achieved. Section **??** will describe a reference implementation of the architecture described here.

The first issue is how the keys, and their corresponding data and/or references to other nodes are stored. As new data is constantly being added to the system, a stack mechanism is employed that allows data that is not frequently accessed to be removed from the system. New key/data pairs are added at the top of the stack, and old data is removed from the bottom. The stack is divided into two sections. The upper section stores keys, a reference to the node from where the key/data originated, and the data itself. In the lower section only the keys and the references are retained. As items move down the stack they move from the first section into the second section, and the data stored with them is deleted (see figure 5.3). Note that in both cases where we might add data to the stack, we will have a key, a reference, and data to add. The first case is where we are forwarding a Data Reply message, in which case we push the key, it's associated data, and a reference to the node where the data originated, onto the stack. The second case is where we receive a Data Insert message. Here we push the key and data onto the top of the stack, and give the source of the message as the reference[7].

A second consideration is how the system is to handle incoming messages resulting from a Data Request, or Data Insert that we have previously forwarded. The method chosen is to employ a callback system, where, on receiving a Data Request or Data Insert message, a 'callback' object is placed in a stack along with the ID of the message. When a message is received this stack is checked for any callback objects corresponding the ID of the received message, and if so the message is handled by that callback object, rather than the default message handling code. It is possible that a message that we forwarded will get lost, and not be returned. The system should not allow such callbacks to persist indefinitely, and thus a "culling" system is employed, not unlike that described in the previous paragraph. The maximum number of callback permitted can be specified, and if that number is exceeded then the oldest callbacks are removed from the system.

[6]It should be noted that there is room for improvment in the behaviour of nodes on receiving Data Insert messages, see section 9.1 for further details.

[7]It may be desirable to use the node that the message is forwarded to as the reference instead. This decision could be made on a node by node basis. See section 9.1 for further details.

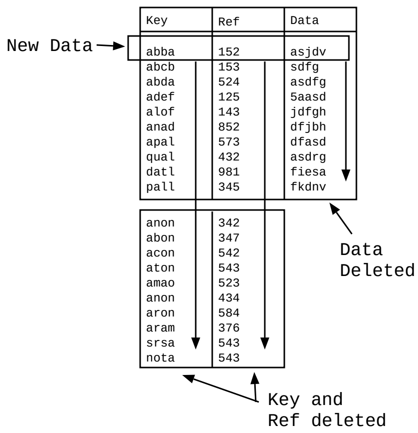


Figure 5.3: Data storage stack within a node

Chapter 6

A simulation

6.1 Design Aims

In the design of the full simulation I hoped to achieve the following:

- Reusable code
  Code in the simulation should be easily transferred into a full working prototype of the network.
- Modular
  The design should allow easy migration between different network types and configurations, for example the type of keys and the methods by which they are compared should be easily changed. Java's object orientated nature helps to achieve this aim.
- Self documenting
  The self documenting features of Java should be fully exploited to aid in any effort to extend the simulation.
- Supports multiple experiments
  The simulation should support a wide variety of possible experiments, and be as flexible as possible in terms of data-measurement.

6.2 Simulation Design

The modules that make up the simulation software are divided into three packages, **Freenet.FullSim**, **Freenet.FullSim.sim**, and **Freenet.FullSim.util**. See figure 6.1 for a diagram of class dependencies between the Freenet.FullSim and Freenet.FullSim.sim packages.

6.2.1 Freenet.FullSim

The classes defined in this package are all sufficiently generic to be used, unchanged, in a final prototype of an Adaptive Network.

6.2.1.1 NetworkInterface

This interface handles interaction between a Node and other nodes in the adaptive network. It defines a method **sendMsg** which allows a node to transmit a Message to another node in the adaptive network.

6.2.1.2 Key

This abstract class represents a Key that may be used to locate information.

6.2.1.3 Message

This class represents a Message that can be sent from one node to another via a NetworkInterface. Specific message types such as **DataRequestMessage** and **RequestFailedMessage** are subclasses of this class.

6.2.1.4 Node

This class represents an individual node in the network. Nodes handle messages passed to them by a NetworkInterface, and send messages back via that interface. Each Node has a DataStore in which it stores keys, data, and references to other nodes. Each node remembers what messages it has forwarded so that it can handle them appropriately if it sees a message with the same ID again.

6.2.1.5 AddressInterface

This interface is the location of a node on a network, for example an IP address. Using a subclass of AddressInterface a message can be sent to any node on the network. Each node has one AddressInterface associated with it.

6.2.1.6 DataStore

A DataStore object handles the storage of keys, data, and references to other nodes. It handles addition of new keys, and automatically handles deletion of data and references (see figure 5.3). Figure 6.2 is a diagram of how objects interact within the simulation.

6.2.2 Freenet.FullSim.sim

This package contains classes that are specific to simulating an Adaptive Network.

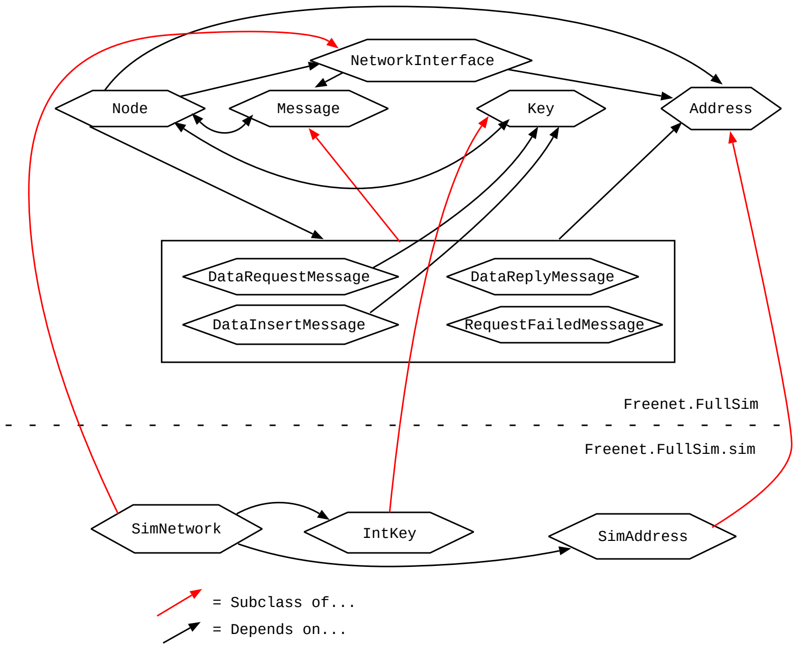


Figure 6.1: Dependencies Between Classes in Freenet.FullSim and Freenet.FullSim.sim packages

6.2.2.1 SimNetwork

The SimNetwork class implements a NetworkInterface (see section 6.2.1.1). A SimNetwork contains several nodes and handles message passing between those nodes. It also allows a user to send messages to any of the nodes, and has facilities for monitoring the behaviour of the simulated Adaptive Network. It works by having an internal FIFO queue of messages, new messages that are sent by nodes are added to the bottom of this queue, and messages are taken from the top of the queue to be sent to nodes. This means that initially several requests could be placed into the network, and then these requests would be carried out in parallel.

6.2.2.2 SimAddress

The SimAddress class extends the Address class (see section 6.2.1.5). A SimAddress is the location of a node within a SimNetwork.

6.2.2.3 IntKey and StringKey

These are subclasses of the Key abstract class which allow the use of integers and strings respectively as keys within an Adaptive Network.

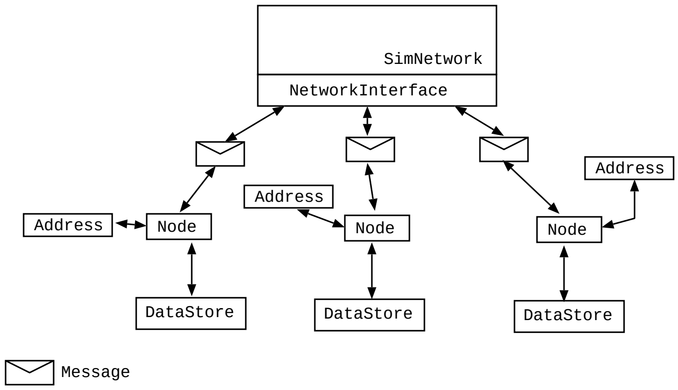


Figure 6.2: Interaction of objects within simulated Adaptive Network

6.2.2.4 Freenet.FullSim.util

This package contains general utility classes.

6.2.2.5 BinaryTree

A binary tree that can be used to rapidly store and retrieve integer-indexed information.

6.2.2.6 LimitedBinaryTree

This class implements a binary tree with limited size. If data is inserted into the tree when it is full, then the oldest data will be deleted. It is used to store callbacks for messages by Node.

6.2.2.7 Fifo

This is a standard First-In-First-Out queue (used to store pending messages by SimNetwork)

6.3 Extending Code for use in Prototype of Adaptive Network

6.3.1 Class re-implementation

The various classes that constitute the simulation have been written to allow a significant amount of code reuse. All simulation specific classes have been placed in the Freenet.FullSim.sim package, where as classes in the Freenet.FullSim and Freenet.FullSim.util packages are also usable in an actual implementation of an Adaptive Network client.

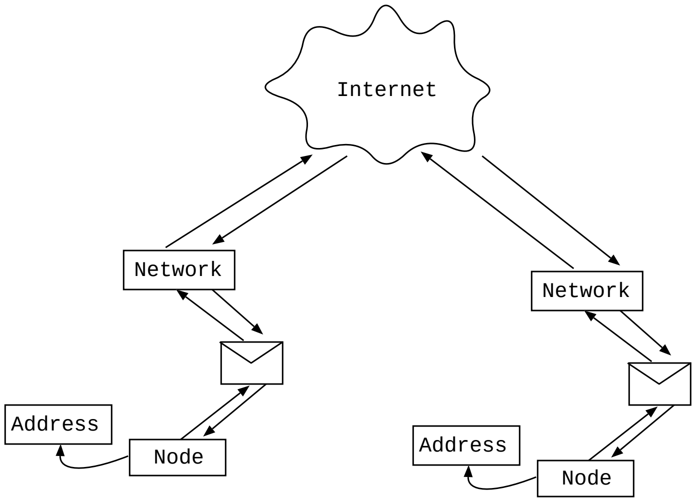


Figure 6.3: Architecture of Adaptive Network prototype

To create a prototype would require re-implementation of the NetworkInterface, and Address classes. In the simulation the NetworkInterface *was* the medium through which all nodes communicated, in a working prototype it would merely serve as an interface to the Internet for a single node, handling message transmission and reception. An Address class must also be created which can represent the address of another node on the Internet. This could consist of a 4 byte IP address, along with a 1 word port number indicating which TCP port the client software is listening to[1]. Such a network interface could be extended to cope with future versions of IP (which have much longer IP addresses).

6.3.2 Communication mechanism

Creation of a prototype would require determination of how messages are transmitted from one client to another. In the previous paragraph use of TCP port numbers is mentioned, implying that the stream-based Transpost Control Protocol would be used to transmit messages between nodes. The alternative is using the unreliable, but fast, packet orientated Uniform Datagram Protocol or UDP. Usage of this protocol would require the incorporation of error handling facilities into the client. Usage of the TCP stream based protocol also allows concurrant forwarding of long messages. Rather than a node waiting until a DataReplyMessage and the potentially lengthly data it carries being transferred completely to a node prior to it being forwarded, a node could "feed" the incoming DataReplyMessage into an outgoing message as it arrives. Through this method, the eventual node receiving a DataReplyMessage could begin to receive it before the node on which the data was found has finished transmitting the message, this would lead to significant improvment in efficiency.

[1]This would allow multiple clients to operate on an individual computer, and also affords more flexibility to the user who may be limited as to what TCP ports they have access to.

6.3.3 Interfacing to the Adaptive Network

It is essential that a convenient method of retrieving data from the Adaptive Network is available if it is to become widely adopted (as would be required for such a system to be useful). One such possibility is to incorporate a HTTP "conduit" into the client. This would allow a standard web-browser to be used to retrieve information. For example, if the HTTP conduit was set to listen to port 8080 retrieving the URL **http://localhost:8080/key=abcdefg** from the computer on which the conduit was running[2] would cause the data with key "abcdefg" to be retrieved from the Adaptive Network. Such a mechanism would even allow information stored on the World Wide Web to refer to information stored in an Adaptive Network, allowing the two information publication mechanisms to be seemlessly integrated. One potential danger with this facility is that such a system may encourage the creation of HTTP "portals" into the Adaptive Network, these would then become high-risk points of potential failure, undermining the purpose of the Adaptive Network. This could be discouraged by making these HTTP conduits refuse connections from any other IP address but their own.

[2]By convention, "localhost" on a Unix computer refers to the IP address of the computer itself

Chapter 7

Experiments

7.1 Changes in network size

7.1.1 Scenario

A network was constructed where integer keys were initially assigned to nodes (each of which was numbered) in ascending order. 10 items of data were assigned to each node, such that node 0 contained data items 0-9, node 1 contained data items 10-19, and so on. In addition, each node is given a reference to the lowest key stored by the node at either side, so for example, node 4 would be given a reference for key 30 to node 3, and key 50 on node 5. The intention is to examine the networks performance once it has adapted, to accelerate this process the network is arranged to ensure that even initially requests will reach destination nodes in a reasonable amount of time. The data associated with each key was an integer equal to the key value squared (although the key-data mapping would have no bearing upon the experiments), see figure 7.1. Each node could store up to 40 data items, and 10 references to other nodes[1].

[1] While this is a very low amount of information that can be stored by each node, limitations on the processing speed and memory of the available hardware imposed these restrictions. These resultsshould be considered a lower-bound as to network performance.

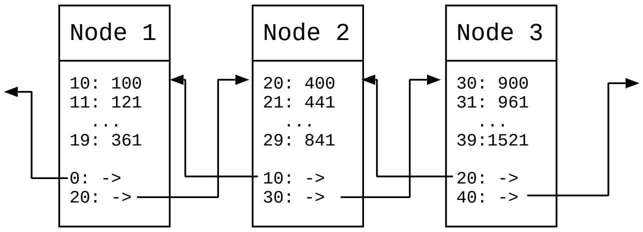


Figure 7.1: Initial network configuration for changes in network size experiments

7.1.2 Information retrieval reliability

Preliminary investigations revealed that if a network was created with data distributed over the nodes in the network, and where each node had minimal knowledge of surrounding nodes, initially most requests did not succeed due to a request timeout. However, as the network adapted the number of requests which succeeded rapidly approached 100%. These experiments are designed to evaluate the effects of different network sizes on an adaptive network's performance.

7.1.2.1 Aims

To determine how network size influences a networks ability to improve the percentage of successful retrievals.

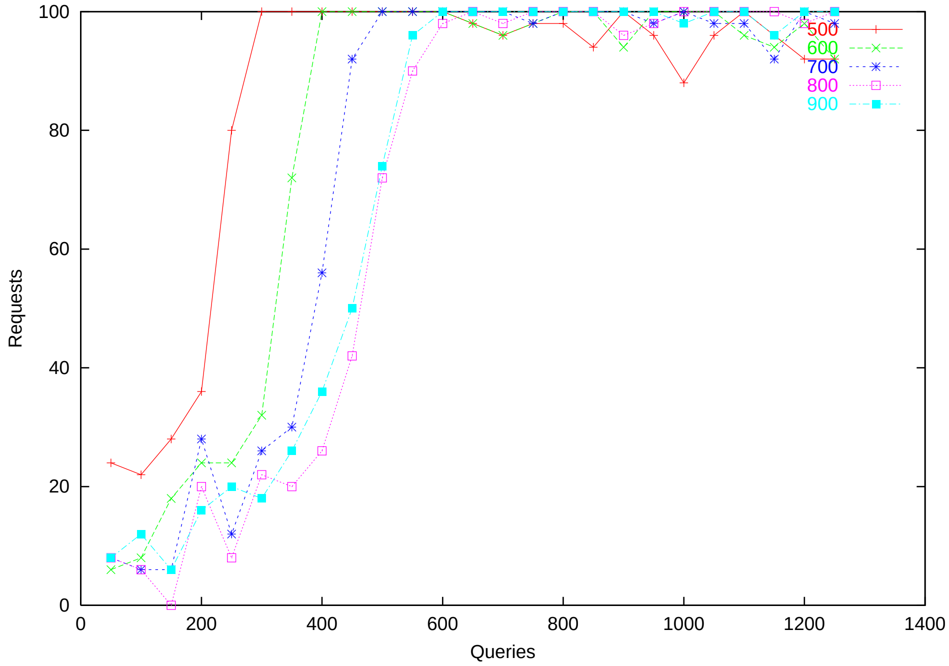
7.1.2.2 Method

Several networks were constructed as described in section 7.1.1 of various sizes. Queries for random keys were sent to random nodes in the network. Every 50 queries the percentage of these queries that successfully returned the data was sampled, each batch of these 50 queries was executed in parallel[2]. A total of 20 such samples were taken.

7.1.2.3 Results

The percentage of successful requests are graphed against the total number of requests since the network was initialised. The results for several different sizes of network are shown (the key for the different network sizes are at the top right of the graph).

[2] See section 6.2.2.1



7.1.2.4 Observations

In the case of all network sizes tested performance rapidly improved until over 95% of requests were successful. Unfortunately due to hardware limitations nodes of comparable storage capacity to those which would be available in an actual implementation of an Adaptive Network could not be simulated, however even these extremely limited simulated nodes retrieved information with reasonable reliability, and this serves as a lower-bound for the performance of an actual Adaptive Network deployed in a real world setting.

7.1.3 Information retrieval time

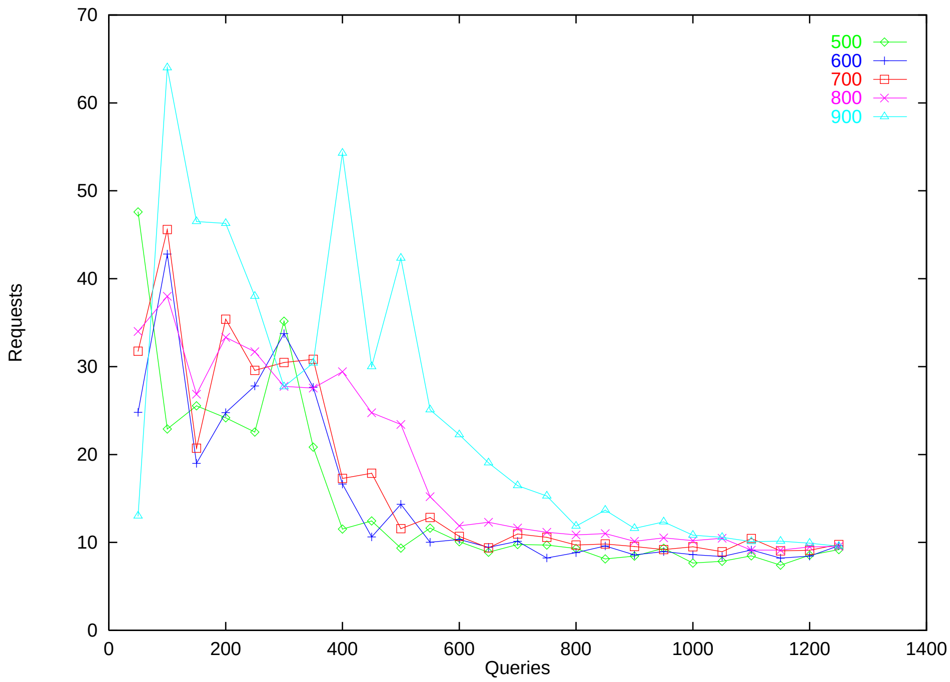
7.1.3.1 Aims

To determine whether the number of requests required to retrieve information decreases as the network adapts to a random distribution of requests.

7.1.3.2 Method

Several networks were constructed as described in section 7.1.1 of various sizes. Queries for random keys were sent to random nodes in the network. Every 50 queries the number of messages transmitted (a value that would approximate the time required to retrieve a piece of information) was sampled, each batch of these 50 queries was executed in parallel. A total of 20 such samples were taken.

7.1.3.3 Results



7.1.3.4 Observations

This experiment demonstrates that as the network adapts its ability to retrieve information improves. One interesting feature of this graph is that once the network has stablised (after about 800 queries) information in retrieved within approximately 10 requests *regardless* of the size of network. This suggests that the number of requests required to retrieve information from a network grows very slowly in proportion to the size of the network, meaning that the system scales extremely well.

7.2 Realistic Scenario Experiments

The experiments described in this section are designed to model the behaviour of an adaptive network in a realistic usage scenario. Initially the network consists of one node, but its size is increased gradually to simulate the natural growth of an adaptive network as new computers are added. After each new node is added new information is added to the network using DataStoreMessages which are sent to random nodes in the network. This information is added in ascending order. A set of requests are then sent to random nodes in the network. To simulate the fact that more recent information placed in a network is more likely to be popular, only the most recent 25% of data to be added to the network will be requested (ie. if the most recent key to be inserted was 400 then only keys 300 to 400 are liable to be requested).

7.2.1 Data Movement during Network Growth

7.2.1.1 Aims

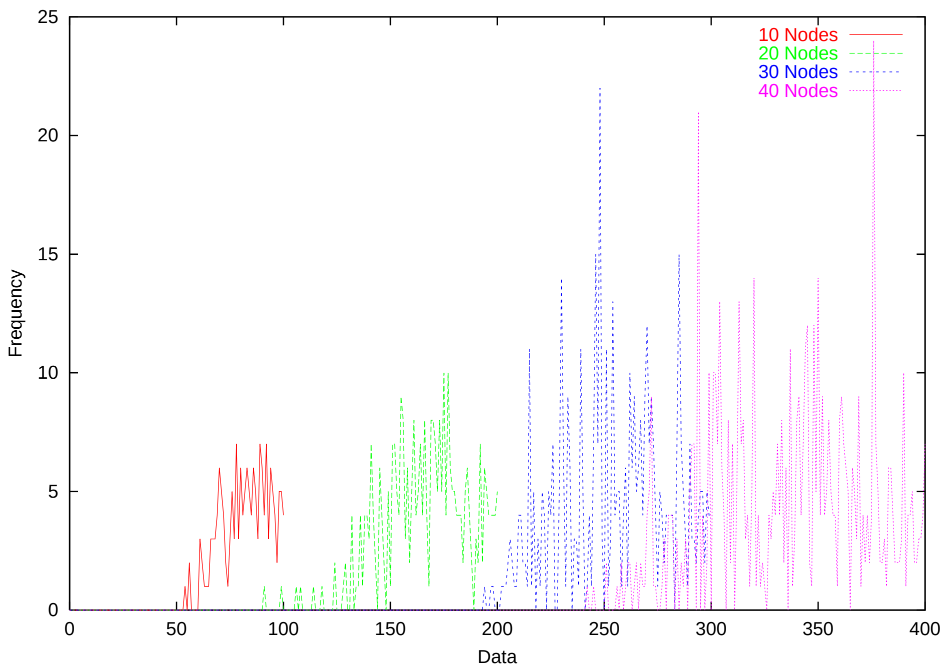
This experiment is designed to show how information is stored within the network, and how rapidly the network removes information that is no-longer requested.

7.2.1.2 Method

The network was started with one node, and the size was increased as described in section 7.2 up to a size of fifty nodes. After each node was added to the network, 10 items of data were inserted at random nodes, then 50 requests were sent to random nodes in the network (for recently added data - see 7.2).

7.2.1.3 Results

After 10 nodes were added the nodes in the network were scanned to count the number of times the data for each key was stored. This is graphed here for the various sizes of network.



Chapter 8

Conclusion

The original aims of the project have been achieved. The Adaptive Network has no element of centralised control or administration. Through merely defining a minimal protocol for communication between nodes, along with reccomendations as to the behaviour of nodes on receiving particular messages, the individual user is free to implement the specifics of how their computer participates in the Adaptive Network. I have created a simple implementation of a node forming part of an Adaptive Network, and demonstrated through experimentation that even this simple implementation is capable of performing reasonably well, and that the assumptions upon which the Adaptive Network depends are valid. It is important to note that due to the wide scope of this project there remains many areas to investigate, and there is much potential for improvment in the operation of individual nodes in the network.

Chapter 9

Areas for further investigation

9.1 Improvements to data insertion

The example implementation handles DataStoreMessages in a simplistic manner, to which many improvements may be made. Firstly, backtracking could be introduced as this would more closely mirror the behaviour of DataRequestMessages as they search for information. Further, rather than depositing a copy of the data at every node, the DataStoreMessage could search the network before selecting a single node where the data could be stored.

9.2 Use of encryption to enhance security

Through use of an encryption system the security of this system could be further improved. For example, rather than using descriptive strings as keys to information, some form of trap-door function which would allow the conversion of a descriptive string into a key. This would make it more difficult to determine what kind of information is being retrieved by various nodes.

9.3 Use of digital signatures to allow updating of information

One shortcoming of the system described at the moment is that there is no option to update information in the network, other than by waiting for it to be removed from the network (which may never happen if it is popular) and re-inserting an updated version. One possibility for allowing information updates is to create a public/private key pair when a message is created and include the public key with the message. Any updates may then be "signed" using the private key, and sent using some form of backtracking search method to the cluster of nodes that store the information. Each node may individually use their public keys to verify that the update is valid before changing their data.

Unfortunately, there is no guarantee that all copies of the data can be updated in this way, however through intelligent design of the update message it should be possible to achieve reasonable performance.

9.4 Generalisation of Adaptive Network for data processing

A longer term, and more ambitious goal would be to determine whether a distributed, decentralised, data processing system could be constructed using the information-distribution Adaptive Network as a starting point. Such a development would allow the creation of a complete distributed decentralised computer

Chapter 10

Future Developments

It is the author's intention to create a fully operational Adaptive Network using the simulation described in this documentation as a starting-point. The client will be written in the Java programming language, and will be developed in an open manner with the cooperation of interested parties on the Internet. It will be released under the GNU Public Licence[1].

[1] GNU stands for Gnu's Not Unix. The GNU Public Licence, when applied to source code, is designed to make it as useful as possible to the public by ensuring that the source code will always be freely available, and can be modified by any person for whatever purpose, provided that the resulting program is also placed under the GNU public licence.

Chapter 11

Acknowledgements

- Dr. Chris Mellish - Supervisor, for his help and guidance during the course of this project.
- Various staff and students in the Division of Informatics for their constructive criticism that allowed me to clarify many of my early ideas
- Numerous people from all over the world for their enouragement and comments on hearing of this project, many of whom have undertaken to assist me in realising an actual working Adaptive Network on the Internet.

Appendix A

Glossary of Java terms

- Class

A Class is a template for an object, in a similar manner to how, in C, a 'int' type is a template for an integer variable.

- Superclass and Subclass

A subclass is a class which embodies the properties of the "superclass", but adds more specific proerties of its own. Fruit might be a subclass of Food, and Apple might be a subclass of Fruit. Food is then a superclass of Fruit, and Fruit is a superclass of Apple.

- Abstract Class and Interface

Interfaces and Abstract classes may not exist in their own right, but may act as superclasses for other classes. For example, a Fruit cannot exist without being an Apple, an Orange, etc etc. We cannot simply have a "fruit", and thus it can be considered an Interface or Abstract Class.

Appendix B

Java Documentation

Java has facilities for automatically generating API documentation for source code. The API documentation for the simulation used for experimentation in this report may be found in HTML format on the world wide web at **:**

http://www.dcs.ed.ac.uk/~iic/4yp/javadocs/packages.html.

Bibliography

[1] "How the Usenet News Protocol Worlds", Jacob Palme- http://www.dsv.su.se/~jpalme/e-mail-book/usenet-news.html

[2] "Introduction to Algorithms", Thomas H Cormen, Charles E Leiserson, and Ronald L Rivest, MIT Press 1990

[3] "The Eternity Service", Dr Ross Anderson - http://www.cl.cam.ac.uk/users/rja14/eternity/eternity.html

[4] "The Java Programming Language Second Edition", Ken Arnold, James Gosling, Addison-Wesley 1998