What can Archives offer the World Wide Web?

Neil Smith
UNIX HENSA,
University of Kent at Canterbury.

March 22, 1994

Abstract

ftp archive sites are well known throughout the Internet community for their wealth of useful information. UNIX HENSA[1] is perhaps one of the better known academic archives in the United Kingdom. In this paper the roles of these archive sites are examined with a view to what they can provide the World Wide Web community. The Web suffers from virtually the same problems as standard **ftp**; slow networks, overloaded serving hosts, and the immense difficulty of finding information spread across the globe. Fortunately, it seems that just as Archive sites have helped alleviate some of these problems for **ftp** users, they should be able to do the same for Web users. In this paper I shall demonstrate how, at the UNIX HENSA ARCHIVE, we have been investigating this possibility.

Introduction

For many years the Internet has had its libraries, commonly known as archives. These machines, often dedicated to the job, are dotted throughout the world and each day spend hours pulling gigabytes of data across the Internet. The whole point of this exercise? To reduce network load and improve response times.

At first this may seem a little contradictory, but the final aim is to try and prevent every user flooding to the same machine to retrieve the same file, across the same network at the same time. Instead the user is encouraged to visit a local archive to retrieve their file. This scheme brings benefits for everyone. The user is able to retrieve the file in question significantly faster and the busy international links are not clogged with simple file transfers, thus leaving more bandwidth for other traffic.

In the past the job has been relatively simple. The quantities of data involved have been large, but at least it has been concentrated in specific places, with good accessibility. The protocol involved, **ftp**, has been cheap, simple and fast. With the introduction and explosion in popularity of the World Wide Web most of these simplifying factors have been lost. The amount of information involved is incomprehensible. This information is scattered over many hundreds of machines, on every network in the world. Some of the most popular machines are severely overloaded. This situation is not helped by the relatively expensive protocol, **http**[2], used to transfer documents across the Web.

The team at the UNIX HENSA ARCHIVE at the University of Kent have been investigating ways in which a typical archive, such as ours, may serve the World Wide Web, in the same way as we serve **ftp** users.

Information and archives on the Internet

The Internet has had plenty of time to evolve. In the beginning anonymous **ftp** was the standard method of making files available to unknown users, external to the local site. This evolution made it feasible to organise these anonymous **ftp** services. It was soon realised that each site providing its own anonymous **ftp** area with its

[1] The Higher Education National Software Archive (HENSA) is a project in the United Kingdom funded by JISC. The aim of this project is to serve academic institutions in the United Kingdom with freely available software for Unix and microcomputer environments. The archive has been split into two parts. At the University of Kent at Canterbury we provide support for Unix platforms, while at the University of Lancaster there is support for microcomputers, primarily, although by no means solely, the IBM Personal Computer.

[2] **http** is the HyperText Transfer Protocol. Details of this protocol can be found at *http://info.cern.ch/hypertext/WWW/Protocols/HTTP.html*.

own material would make it difficult to find and catalogue the information available. The answer to this problem was to provide archives; machines dedicated to the task of serving files via anonymous **ftp**. These archives collect together material from other anonymous **ftp** areas scattered through the Internet and present it in a single location.

The job of the archive maintainers is to keep the archives up-to-date and to try and organise them in an orderly fashion. They are aided by the nature of the material that they are trying to arrange which consists solely of simple files or directory trees. These files have no cross-references and are not dependant on their location within the world. For example, the copy of the latest version of the GNU C compiler on the UNIX HENSA ARCHIVE is identical to that on the Uunet archive. By transferring the file from one location to the other we have not changed the way in which it interacts with the anonymous **ftp** protocol. These simplifying properties are not present in a large proportion of the information available on the Web. Does this stop us providing Web archives?

Can we mirror the Web?

The technique of taking an exact duplicate copy of a remote anonymous **ftp** area and presenting it within a local anonymous **ftp** area is known as mirroring, shadowing or tracking. In principle the technique is simple. A list of all the files available on the local machine can be made. Each day this list is compared with the list of available files on the remote machine. Those files which appear on the remote machine, but not on the local machine are copied across and made available locally. Those on the local machine but not the remote are deleted. Of course the intricacies and details make the whole process more complex than this, but the principle remains simple [3]. Could we not merely take the technology that we have developed and apply the same principle to the World Wide Web?

Unfortunately the answer would seem to be no. The Web has exploded in size[4]. Its growth and acceptance have been beyond any possible dreams. This growth has largely been anarchic with every user who has the capability and desire setting up a Web server on their machine. This has resulted in a very wide, but shallow spread of information which will then typically be infrequently accessed. The format of the information within the Web has been specifically designed to allow documents to cross reference each other and this cross reference information is kept within the document itself.

A strategy that we might adopt if we were to attempt to mirror the Web would be to nominate some starting point and expand from there to make a copy of the whole, or at least selected parts, of the Web. As each document is copied from its home site it would be scanned for cross references to other documents. These cross references might then be rewritten to refer to a copy of the same document in the local mirror. Each day the local mirror would could be traversed in order to find out which documents need to be updated.

Mechanisms exist within the **http** protocol to allow us to see whether the local copy of a document is out of date with respect to the copy at the originating site and there would be no need for archive administrators to provide much additional organisation for the documents as one would hope that the original authors would do this.

So, everything would seem to be in place to allow the archives to provide mirrors of interesting, or popular, parts of the Web. Unfortunately there are some problems which, while they do not totally prevent attempts at mirroring the Web, do make the utility of the resulting copy questionable.

Unlike files in an anonymous **ftp** area, documents on the Web are not necessarily static. When you read a specific document across the Web you are not necessarily reading a plain file, you may be reading the output of some arbitrary executable, or the result of filling out a form[5]. In addition to the HTML documents normally associated with the Web the **http** protocol also encompasses the **gopher**, WAIS, NNTP and **ftp** protocols. How would this information be represented in a mirror? The executable file is certainly not a visible entity within the Web, and a WAIS search may be traversing databases which, again, will not be retrievable entities within the Web.

Another problem with the simple-minded copying of large parts of the Web is the deficiency of **http**. **http** is a one shot protocol. For each document that we want to retrieve we have to make a connection to the host serving that

[3] The standard tool for mirroring anonymous **ftp** sites onto local machines is Mirror, written largely by Lee McLoughlin of Imperial College, London. The package is available as *ftp://src.doc.ic.ac.uk/packages/mirror/mirror-2.3.tar.gz*.

[4] The World Wide Web Worm, (*http://www.cs.colorado.edu/home/mcbryan/WWWW.html* has recently reported finding more than 100,000 multimedia objects available on the Web. This growth from nothing has all happened in the last nine months.

[5] This capability is provided by the Common Gateway Interface, see *http://hoohoo.ncsa.uiuc.edu/cgi/* for full details of this interface and how you can use it in your server.

document. If we require more than one document from a particular server we have to make multiple connections. This is in contrast to the **ftp** protocol where a single connection allows us to retrieve multiple files. When accessing busy servers across busy networks the connection start up time is a significant factor and places a significant load on the serving host if many documents are going to be retrieved in quick succession. Even if we do not need to retrieve a document to replace the copy that we have in the mirror we have to make a connection to determine whether the document at the originating site is newer than the local copy. That is at least one connection start up and shut down for every single mirrored document.

A problem with the standard method of **ftp** mirroring, that any mirror of the Web would inherit, is the bandwidth wasted to obtain files that nobody is going to access. If a document on a remote server is changing rapidly then this document may be mirrored each night onto a local host. It is entirely feasible that during each of those intervening days that document will not be read from the local host. This renders each of the transfers on the previous days redundant and wasteful of bandwidth.

If the standard **ftp** mirroring techniques cannot be applied to the Web as they are, is there any chance of improving accessibility to documents being read across slow network connections or from loaded servers? Fortunately recent work by the developers of Web servers and clients has led to a technique which promises to give us at least as much as the standard **ftp** mirrors.

Caching Proxy World Wide Web Gateway

At the UNIX HENSA ARCHIVE we are been using experimental software developed at Eindhoven University of Technology to provide a Caching Proxy World Wide Web Gateway. To the end user of our services it appears that they are talking to their normal World Wide Web client in the standard way, and accessing the remote servers as they always have done. In fact, behind the scenes, a lot is going on to ensure that the user receives their document as quickly as possible and does not waste precious bandwidth.

The title above describes the technique used quite well. As the user navigates the Web, their client, for example Mosaic or Lynx, knows that it must not retrieve the selected document itself, but must instead use the proxy gateway. The client makes a request of the proxy gateway and the gateway obtains and passes on the document to the client. The advantage of this extra level of indirection is that the proxy gateway can, in addition to passing the document on to the client, keep a local copy. If another client requests the same document, the chances are good that a connection to the originating server will not have to be made. Instead the document can be served out of the local cache. If the cache scheme used dictates that document must always be up-to-date then a connection will have to be made to check the date of the file, but in most cases this will not result in the whole file having to be transferred again.

The proxy gateway at the UNIX HENSA ARCHIVE has been running for a number of weeks. This service is open to anyone who wishes to use it, although we imagine that the users who will benefit most will be those in the United Kingdom. The sections below detail how this service operates, how it addresses the problems that a naive **ftp** type mirror would introduce, describes the current deficiencies and proposed solutions to some of the problems that still exist.

Lagoon - The working details

At the UNIX HENSA ARCHIVE we are currently using version 0.11 of the Lagoon[6] Web Caching software. We use this in association with version 1.1 of the NCSA **http** server. The Lagoon software has two modes of operation.

Lagoon as a Cache

Lagoon can act as a simple cache for World Wide Web documents. In this mode of operation the end user must first prefix the URL[7] of the document that they are going to access with the URL of the cache. For instance,

[6] Lagoon is the World Wide Web caching software. Details can be found at *http://www.win.tue.nl/lagoon/*.

[7] Uniform Resource Locators are the strings used by the Web community to unambiguously identify the resource available on the Web. A description of the syntax of a URL can be found at *http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html#URL*.

suppose that you wish to access the document with URL

http://www.ncsa.uiuc.edu/demoweb/demo.html [8]

If you wished to go via the cache at the UNIX HENSA ARCHIVE then you simply need to add the following text onto the front on the original URL.

http://www.hensa.ac.uk/mirror/

This gives the URL below which identifies the UNIX HENSA ARCHIVE as the resource location, and requests that the archive fetch the original document through the cache.

http://www.hensa.ac.uk/mirror/http://www.ncsa.uiuc.edu/demoweb/demo.html.

In this mode of operation Lagoon will rewrite all the cross references within the document that it returns to the end user. This means that the user does not have to specify the UNIX HENSA cache URL a second time. Once the user has entered the cache they cannot leave unless they specify a URL that deliberately avoids the cache.

The advantage of this scheme is that no support is required from the user's client. Any client that can browse the Web can use Lagoon in this way to retrieve its documents via a cache.

The main disadvantage of this simple caching scheme is that all the links within the document returned to the client must be rewritten in order that future requests go through the cache. This is an error prone task which then makes the URLs within the document that much longer. This rewriting also takes machine time and requires disk space on the caching machine. The machine time required for the rewriting of cross references should be relatively small once the cache matures, but the extra disk space required could be considered as a significant factor. In order that the cross references do not have to be rewritten each time a document is requested, the documents with the references already rewritten are themselves cached separately. The original document from the originating host has to be kept for use with the **http** proxy facility desribed below. This does not result in a doubling of disk-space as binary files, which are often the largest, are not duplicated, but experience with the UNIX HENSA cache shows an increase in disk requirements of about one quarter.

Lagoon as a Proxy Gateway

Using Lagoon as a Proxy Gateway avoids both the problem of link rewriting and extra disk-space usage. However support is required from the Web browser being used[9]. In this mode of operation the documents are served to the requesting client exactly as they are received from the originating server. The links within the documents are not rewritten. Instead it is up to the client to recognise that it must not fetch the document itself, but must make a request via the proxy gateway. Currently, the standard method of communicating this fact to the Web browsers is to use the environment variable *http_proxy*. For example, to use the proxy cache at UNIX HENSA the variable would be set to *http://www.hensa.ac.uk//proxy/.*

This scheme does not have the disadvantages of the simple caching scheme mentioned above. The document does not have to be altered before it is passed to the client, and therefore, additional copies to not have to be kept.

How does this solve the problems of a naive mirror?

Earlier, some of the problems that would be experienced if we were to naively mirror parts of the Web, in the same way as we mirror **ftp** sites, were discussed. The idea of a proxy server solves a large number of these problems.

When using the proxy gateway the end user decides which documents they want to read. There is no requirement for intervention by the archive maintainers. This solves the problem of a user accessing an **ftp** archive and finding

[8]The NCSA server, based around four co-operating machines, is well known for its high load.
[9]Currently X Mosaic version 2.2, Lynx version 2.2 and Emacs-w3 are known to offer this support.

that the file they are looking for is not archived at that site. With the proxy gateway this problem does not exist. It may be that the document is not within the cache, but at least the end user will be able to read it without having to close the connection to the caching server and making an additional connection to the originating site themselves. In future, when they come to read the same document again it is likely to have been kept in the cache. This uncontrolled accessibility means that the lack of organisation within the Web is not a problem, at least not for the archives.

In future, the proxy gateway is likely to become the standard for Web caching. The simpler caching scheme offered by Lagoon is likely to become less and less used as more and more Web browsers come to support the proxy gateway service. With the acceptance of this scheme the rewriting of the links within Web documents becomes unneccesary. This can only be beneficial as the rewriting of URLs is error prone and makes unneccesary demands on machines which may already be heavily loaded. This rewriting would have been required by any scheme using a naive mirror.

Dynamic documents, that is, those which are dependant on some input or different each time they are read, can be handled by the proxy gateway in a sensible way. As long as the document can be recognised as being dynamic the cache can be instructed not to bother keeping a local copy. Currently, it is the recognition of which documents are dynamic and which are not which causes most problems. This is discussed in the next section.

When using the proxy gateway, the fact that **http** is a one shot protocol is not such a serious deficiency. The number of connections that will have to be made across crowded networks has already been reduced by the fact that the proxy gateway is caching documents. Depending upon the cache refresh scheme used, it is possible that the only connections that still have to be made are between machines which are local to one another, (the end user and the proxy gateway), or well connected on the network, (the local proxy gateway and the remote site or cache). If the document required is not within the local cache, or is determined to be out-of-date, (either by arbitrary heuristic rules, or by actively checking the fact), then at most two connections will also have to be made to the originating site. Two connections for a document may be considered a significant load, unless it is compared to the requirements for a naive mirror. In the mirroring case at least one connection has to be made for every single document in the mirror. For the proxy gateway, a connection may not have to be made at all, (depending on the cache refresh scheme), and if connections are required, they are only for a document which is sure to be read.

The basic method of operation of the proxy gateway gets around the problem of wasting bandwidth by not fetching files which are never going to be accessed. The fact is that the proxy server will only fetch a file that is actually requested. This method certainly puts less load on the network and the remote machine, but has the disadvantage that if a document has to be read from the remote machine it is likely to be during prime time on the local machine. At least normal **ftp** mirrors have the saving grace that they operate during the non prime time periods during the day.

It seems that once the proxy gateway protocol has been accepted and implemented by all the **http** servers and clients, network load could be reduced dramatically. Unfortunately it is not quite as simple as described above. There are still some problems which remain to be solved.

Problems that still exist, and possible solutions

The most serious problem with the proxy caching system can take two forms. The first of these, described above, is the problem of knowing whether a document is static or dynamic. There is no point caching a dynamic document as by its very nature it is intended to be different each time it is read. If we cache the document and pass that onto subsequent users we have broken the intended behaviour of the remote system. The second problem is to know the lifetime of a static document. That is, to know when it has changed and when our cache should be refreshed. These problems can be considered as one and the same if we consider a dynamic document to be the same as a static document with a zero length lifetime. That is to say that if the document is kept in the cache, it is immediately marked as being out of date, (ideally, in this situation it should not even be kept in the cache).

A number of solutions have been proposed to solve this problem. Unfortunately, while many of them are being actively discussed, none of them seems to have been accepted as a standard in the Web community. Some of the more popular suggestions are described below.

The Lagoon caching proxy gateway determines that a document is out of date based on heuristic rules specified by the cache maintainer. When a file is requested and found in the cache, Lagoon tries to match the URL against a set of patterns. The first of these patterns that matches the URL specifies how to decide whether the document is out of date and needs to be refreshed. For example, the cache configuration file at the UNIX HENSA ARCHIVE looks a little like this:

```
r .gif 1w       # Refresh gifs after a week

r .uk: 1d
r .uk/ 1d       # refresh UK pages after one day

r * 2d          # refresh everything else after two days
```

With the comments next to the rules this is fairly self-explanatory. The archive maintainer has decided that GIF files are likely to remain static and should only be refreshed once they are a week old. URLs from a United Kingdom domain will be refreshed once they are a day old and everything else will be kept in the cache for a maximum of two days.

If a standard could be devised to specify whether a document is dynamic or static within the URL this scheme would probably be good enough for most situations. It has the distinct advantage that no connection has to be made from the proxy gateway to the remote host if the document is considered to be up-to-date within the cache. The cost of this is that a rapidly changing static document at a remote host may remain out-of-date within the cache for two days before being refreshed. If it is a GIF file that is changing rapidly, a weather map for example, then the situation is even worse.

Other schemes guarantee to always give an up-to-date copy of the document, but have the overhead of always having to make a network connection. Some of these schemes have been partially implemented while others are still being discussed.

Within **http** there is a method by which a client can request that the server provide information about a specified document [10]. If a remote server supports this method then one field of the information that it should return is the last modification date of the file. With this information the proxy gateway can decide whether the remote version is newer than the version currently in the cache. If this is the case then the document is retrieved and replaces that in the cache.

In the worst case this method will make two connections to the remote machine for each connection that the gateway receives itself. The first of these to determine whether the document is out of date and the second to fetch it if it is.

An alternative request method has been proposed. This method allows the proxy gateway to give the remote server a date. The remote server then compares this with its copy of the document. If the document is newer than the date provided by the gateway then the remote server responds with the new document. If the gateway has an up-to-date copy then the response reflects this fact. This method has been described as a conditional GET[11]. This places the responsibility for recognising dynamic documents with the originating server. This method has the advantage over the HEAD method described above, in that, even in the worst case, only a single connection needs to be made to the remote machine.

The final scheme that will be discussed here is to provide information within the document itself that describes when it will become out-of-date. This information will not be made visible to the end user reading the document, but the proxy gateway will be able to find it. This scheme would work in a similar way to the expiry date on USENET news articles for example. Just as a news article can have an expiry date so could a Web document. Once that date has been passed a new copy would have to be fetched.

Unfortunately this final scheme makes rather a large demand on the author of the document. For each document written some judgement has to be made saying how long the document is valid for. While this may be easy for each individual document, specifying this information for every document on the Web would be time consuming. In general the consensus seems to be that anything requiring human intervention on such a large scale might as well be ignored.

[10] This method is called the HEAD method as it requests header information for the specified document.

[11] GET is the method normally used to fetch a document. In this instance it is regarded as conditional because the server will only return the document on the condition that its copy is newer than the date supplied by the proxy gateway.

Any of the above cache refresh schemes could be implemented. Which one the Web commmunity will decide upon is not yet known, although the conditional GET method looks as if it will become part of the **http** protocol. If we require documents guaranteed to be up-to-date then we will have to put up with the network overheads involved. In any case this problem is not insurmountable and caching should make the Web even more pleasant to use.

How should we use caching proxy gateways to improve the Web?

The simplest form of caching proxy gateway is, as we shall see in the next section, certainly becoming more and more accepted on the Web. However, by itself, it is certainly not the ultimate solution for the problem of rapid document distribution and retrieval. It goes a significant way towards making things faster and smoother for a large number of users. It even manages to achieve this while avoiding some of the unneccesary overheads of earlier systems solving similar problems, that is, the **ftp** mirrors. But with a little organisation things could be significantly better.

Currently most of the discussion has been based around the idea of a Web browsing client using a single proxy gateway to fulfil all its **http** requirements. What may be a better solution would be to have proxy gateways fulfilling specific roles, and the clients going to whichever gateway is likely to give the best response. In fact, the task of deciding which gateway to visit does not necessarily have to be placed on the client. It could be shifted to the proxy gateway itself. Suppose, for example, that an end user in the United Kingdom follows a link to a document in the United States. If the user is at a site with a proxy gateway and a small dedicated cache, then it is likely that their client will communicate its request to this local gateway. If this local gateway cannot fulfil the request itself, instead of immediately sending a request to the United States it may be directed towards another proxy gateway recognised for its large cache of documents from the United States, a national archive site such as UNIX HENSA for example. Only if this national cache doesn't have the requested document will a connection be made to the United States. There are several analogies to this situation already in existence on the Internet.

The **archie** servers have the enormous task of cataloguing every file available on the Internet by anonymous **ftp**. Of course, each **archie** server does not scour the whole world for every file itself. Instead, an agreement exists between servers such that they can swap information about the files that they have catalogued. One server is responsible for a certain portion of the globe, the others copy the information for this region in exchange for that which they can provide.

There is no single point from where USENET news originates. Instead the articles are distributed from machines high in the hierarchy of newsfeeds to those lower down. It is not up to one machine to serve all articles[12].

Anonymous **ftp** archives have mirror sites in order to reduce the load to acceptable levels. These mirror site are spread around the globe so that it is only in rare cases that a user will have to use international networks.

The situations described above show how, for other network services, the load is distributed across many machines in many parts of the world. Currently, this does not exist on the Web. It is for this reason that we should not be surprised that short documents take, literally, minutes to arrive from popular sites.

Some form of co-operative caching has to be put into place to lighten the load on these popular machines. We have already seen the crippling effect of having a single source of popular information. Now that the proxy gateway is in service at UNIX HENSA it is a rare day when any document takes more than a few seconds to arrive.

Unfortunately the information on the Web is not classifiable by subject. This rules out a scheme similar to that used for **ftp**. (Just because a client specifies UNIX HENSA as their proxy gateway does not mean that they are going to be accessing Unix related material). However, we can guarantee a classification by location. Each document has associated with it, its URL. This should allow us to put into place a scheme, similar to the **archie** database exchange scheme, where specific archive sites become well known for their effective caching of a large number of domains. With these caches in place a URL would no longer specify where to fetch a document from, but rather the last resort if the document is not found in some, unspecified, closer cache.

The order in which caches are searched for the requested document should not have to be specified by the end user.

[12] On close inspection we see that this is exactly contrary to the situation that we have on the Web at the moment. On the Web, there is only a single point of distribution for any particular document. Users reading a popular document from a loaded server will already be painfully aware of this fact.

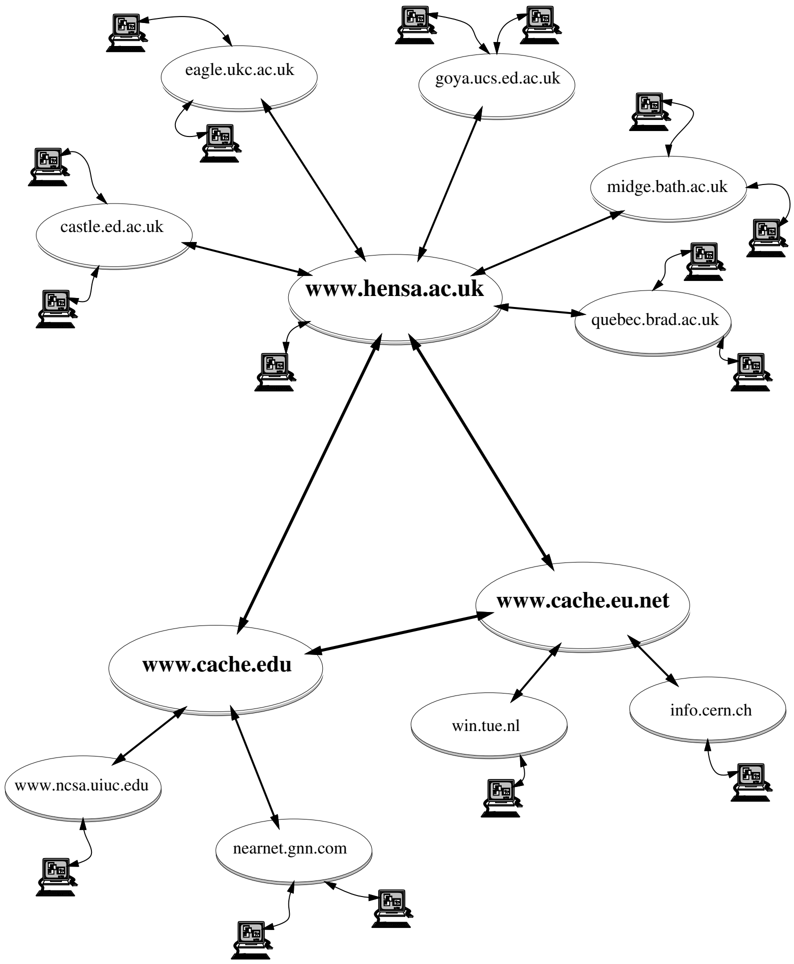

Figure 1: An example of a co-operative cache topology.

Two schemes have been suggested to resolve this problem. The first described here requires no extra supporting framework, while the second is a more comprehensive scheme, but requires quite a lot of additional work.

The first scheme relies on each cache knowing where an end user should try next for the document that they are looking for. This information would be configured into the cache by the cache maintainer. Taking a previous example, the cache configuration at a small site would redirect all failed requests to a larger national cache. If the national cache could not satisfy the request then the configuration would redirect the request to other international caches dependant upon the domain specified in the original URL.

In figure 1 we can see how this scheme might work. Suppose a user on the host *eagle.ukc.ac.uk* wishes to read a document on the server at *www.win.tue.nl*. Their client may be instructed to use the proxy gateway on *eagle.ukc.ac.uk*, but as this is a small cache it is unlikely that the document will be available. The configuration for the cache on *eagle.ukc.ac.uk* is quite simple. It knows that any request that it cannot fulfill itself should be routed to the national cache at *www.hensa.ac.uk*. If the *www.hensa.ac.uk* cache does not contain the document, then it will be configured to recognise URLs from domains within Europe and forward the request to the (imaginery) cache *www.cache.eu.net*. If this cache does not have the document then it will be configured to forward requests for domains within Europe to the site specified in the URL.

In the worst case, that is when the document in not in any of the intervening caches, the transfer time is likely to be considerably longer than if the client had gone straight to the server at *www.win.tue.nl*. However, the hope is that this worst case will not arise often and that many fewer connections will have to be made. Within a short time the more popular documents should migrate towards the areas of the world where there is a large interest in their contents.

The second scheme places an extra level of mechanism between the end user and the information that they wish to fetch. Instead of specifying URLs the user would have to specify URNs[13]. The URN specifies which document is required and a URN server makes some mapping between the specified URN and the locations of documents matching that URN. It would then be up to the client program to decide which of these copies is the closest and fetch the document.

It has been suggested on the *www-talk* mailing list[14] that a scheme similar to the DNS[15] be used to resolve the problem of mapping from URNs to URLs. Each document would have an entry within the distributed database describing its starting location. When a document is copied to another site and made available an entry is added to the database to indicate this fact. When a client makes a request for that document, they will receive a list of locations and can then choose where to fetch it from. If a single database entry for each document seems a little excessive, (remembering that the World Wide Web Worm found more than 100,000 multimedia objects on the Web), then an alternative suggestion is to have a database entry for each server, with addition entries for gateways designated as caches for the original server.

At the moment it would seem that the short term solution to this problem would be to use the first scheme described above. This can be implemented immediately and will deliver benefits straight away. The URN scheme is required anyway, but needs considerable additional work. Once the framework arrives there should be little problem transferring to this from the cache configuration scheme.

The performance of a real proxy gateway

The figures and tables within this section describe the UNIX HENSA proxy gateway as it currently stands. The cache contains 7554 documents. 4445 of these have duplicates with the cross references rewritten for use by Lagoon in its caching mode of operation. In total these occupy 184 megabytes of disk space.

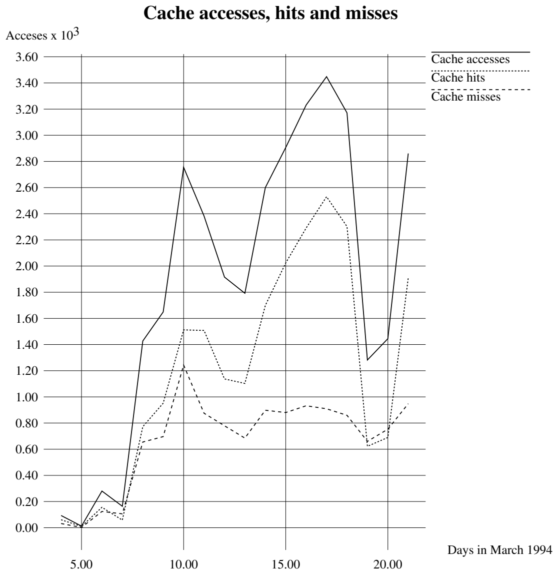

Figure 2: Cache accesses, hits and misses.

Figure 2 shows the number of accesses that the UNIX HENSA cache has served in total, along with the number which have been served from the cache, and the number where the remote site had to be visited. The popularity

[13]A Universal Resource Name is an identifier used to uniquely identify a resource and provide persistent naming for networked objects, no matter the current location(s) of the object. The full Internet Draft defining URNs can be found at *ftp://unix.hensa.ac.uk/uunet/inet/internet-drafts/draft-ietf-uri-resource-names-01.txt.Z.*

[14]To subscribe to this list you need to send email to *listmaster@info.cern.ch*, with the body of the message containing the string `subscribe <list> <your name>`. There are two lists, *www-announce* for announcements relating to the Web and *www-talk* for discussion on Web related matters.

[15]The Domain Name Server is a distributed database of machine and network names. Site register themselves with the DNS and are then responsible for responding to name queries within their domain. Some caching is used within the DNS to improve performance. This caching can result in stale information. A time-to-live is associated with each name and can be used to reduce the chance of stale information.

of the cache has gradually risen and has peaked at about 3,500 accesses each day. The low number of accesses at the weekends can be seen quite clearly.

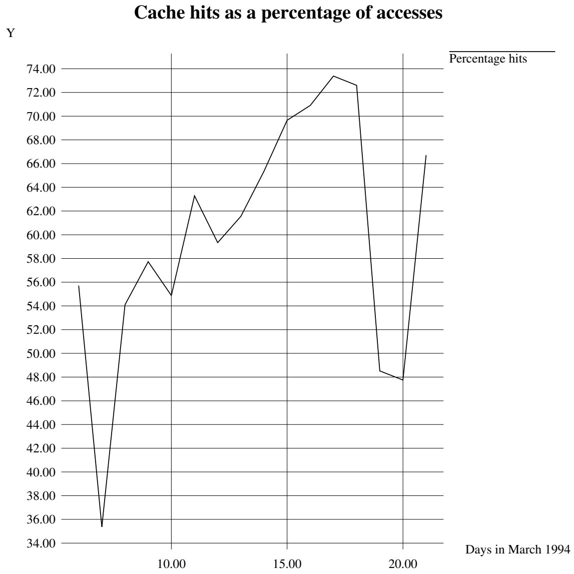

Figure 3:

Figure 3 shows how the number of cache hits has increased as the cache has been used. Cache hits during the week remain quite consistent at around 60 percent of accesses. At the weekend, with fewer accesses but the same chance of a document within the cache being regarded as stale, the hit rate drops noticeably.

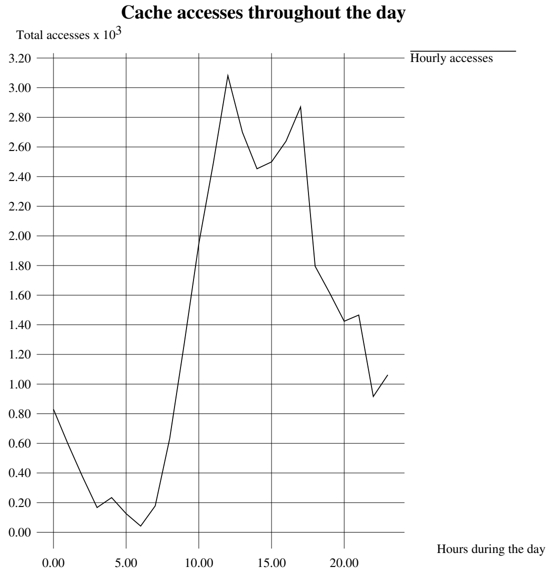

Figure 4: Cache access throughout the day.

Figure 4 shows how the cache requests are distributed throughout the day. While the UNIX HENSA cache was announced globally it seems that users are using their better judgement and, reinforced by the information in figure 5, it can be seen that the vast majority of requests come within United Kingdom working hours. Figure 5 includes requests from machines at the UNIX HENSA home site, (The University of Kent at Canterbury). This data is valid as it represents requests from the University community and a real saving of network bandwidth. This is not just test data.

Figures 6 and 7 show how the documents fetched through the cache are distributed around the world. The reputations that some sites have for being popular are confirmed here. Perhaps many users of the cache are

Hostname	Accesses	Hits	Misses
eagle.ukc.ac.uk	1641	755	881
castle.ed.ac.uk	1394	944	447
gos.ukc.ac.uk	735	337	398
geovax.ed.ac.uk	665	572	93
midge.bath.ac.uk	637	346	289
ewe.dcs.ed.ac.uk	623	353	270
163.1.129.100	616	371	245
raven.ukc.ac.uk	596	364	231
stork.ukc.ac.uk	595	288	306
goya.ucs.ed.ac.uk	538	298	240
swallow.ukc.ac.uk	529	287	240
festival.ed.ac.uk	527	349	177
unix.hensa.ac.uk	451	286	165
spinoza.jesus.ox.ac.uk	413	228	185
thorbb.dcs.aber.ac.uk	409	313	96
lucy.ukc.ac.uk	340	183	157
bunnahabhain.hgu.mrc.ac.uk	334	180	154
snipe.ukc.ac.uk	314	204	107
quebec.brad.ac.uk	298	231	66
falcon.ukc.ac.uk	297	173	124

Figure 5: The hosts making most requests via the UNIX HENSA cache.

accessing these site for the first time, as in the past they have been so heavily loaded as to make interactive use impractical.

After a number of weeks, usage of the Web cache at the UNIX HENSA ARCHIVE seems to have stabalised. On a typical working day we expect to serve at least 60 per cent of requested documents out of the cache. This represents a considerable reduction in network and Web server load.

Conclusion

The Web cannot continue to grow in popularity as it has done over the first year of its existence. If we are to have any hope that the networks and World Wide Web servers are going to be able to cope with the increased demand then we need to place some organisation over the anarchy that has gone before. Any relatively frequent user of the Web has experienced the problems associated with reading documents available only from a single source. The experience gathered while running a Web cache at the UNIX HENSA ARCHIVE has shown that with more work this technique should solve many problems. The large archives can make their contribution to the Web by providing national Web caches.

Uniform Resource Locator	Accesses	Hits	Misses
http://rs560.cl.msu.edu:80/weather/	1076	1068	8
http://info.cern.ch/hypertext/WWW/Icons/WWW/Virtual_Library.gif	547	542	5
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/StartingPoints/NetworkStartingPoints.html	420	411	9
http://nearnet.gnn.com/gnn/graphics/Lgo.gif	399	394	5
http://nearnet.gnn.com/gnn/graphics/home-nav.gif	397	393	4
http://nearnet.gnn.com/gnn/graphics/Lreturn.gif	327	322	5
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/whats-new.html	325	311	12
http://nearnet.gnn.com/gnn/graphics/mkt-nav.gif	293	290	3
http://nearnet.gnn.com/gnn/graphics/res-nav.gif	287	283	4
http://nearnet.gnn.com/mkt/travel/graphics/TRC-nav.gif	278	274	4
http://nearnet.gnn.com/mkt/travel/weather.html	259	252	7
http://rs560.cl.msu.edu:80/weather/weathericon.xbm	243	235	8
http://rs560.cl.msu.edu:80/images/msuUCG.gif	235	231	4
http://rs560.cl.msu.edu:80/weather/smallgmsvis.gif	234	230	4
http://rs560.cl.msu.edu:80/weather/smalld2.gif	234	230	4
http://rs560.cl.msu.edu:80/weather/smallatlanticir.gif	233	229	4
http://rs560.cl.msu.edu:80/weather/smallworldir.gif	222	219	3
http://rs560.cl.msu.edu:80/weather/smallarctic.gif	218	215	3
http://rs560.cl.msu.edu/weather	215	210	5
http://krakatoa.jsc.nasa.gov//Images/	215	212	3

Figure 6: The most popular URLs read via the UNIX HENSA cache.

Hostname	Accesses	Hits	Misses
nearnet.gnn.com	4801	3993	806
rs560.cl.msu.edu	4261	3973	287
www.ncsa.uiuc.edu	4144	3133	998
info.cern.ch	1916	1466	443
sunsite.unc.edu	1191	802	388
www.ucs.ed.ac.uk	923	664	257
www.tue.nl	770	696	74
wings.buffalo.edu	576	387	189
www.cm.cf.ac.uk	488	168	320
hoohoo.ncsa.uiuc.edu	453	246	207
krakatoa.jsc.nasa.gov	314	273	41
info.er.usgs.gov	306	86	220
tns-www.lcs.mit.edu	280	194	86
ivory.nosc.mil	274	163	111
white.nosc.mil	267	217	50
nutmeg.ukc.ac.uk	235	43	192
www.rpi.edu	229	105	124
www.cern.ch	220	83	137
www.eeb.ele.tue.nl	202	108	94
cui_www.unige.ch	200	88	111

Figure 7: The most popular remote sites accessed via the UNIX HENSA cache