Evaluation of Hierarchical Webcaching and Satellite Distributed Caching

Roman Pletka & Oliver Stoeckle

Assistant: Pablo Rodriguez
Prof. Ernst Biersack

Institut EURECOM
2229, route des Crêtes
06904, Sophia Antipolis Cedex, FRANCE

December 16, 1998

1 Abstract

The World Wide Web is growing exponentially and already accounts for a big percentage of the traffic in the Internet. Often popular Web servers are overloaded, hot documents travel many times across the same congested links, and receivers experience slow response times. Cache hit rates can be significantly increased by having caches cooperate. In this report we extensively analyze the log entries of the Eurecom and other Squid caches [[1]] in order to show what hit rates might be achieved with cooperating caches. We also discuss how to chose a parent cache out of several sibling caches based on ping and download round trip times.

2 Introduction

The growth of the World Wide Web is overloading popular servers and increasing the traffic in the network and the response times to the clients. Web caching is being extensively used in the Internet to alleviate these problems. Web caching works as follows: when a client issues a request, the request is first directed to the cache. If the document is found in the cache the document is delivered directly to the client. If the document is not hit in the cache, the document is fetched from the origin server and a copy is placed in the cache. Further requests for the same document are satisfied from the cache. Requests not satisfied in the cache are called misses. Misses can be classified into:

- First-Access: Misses occurring when accessing documents the first time.
- Storage Capacity: Misses occurring when accessing documents previously requested but discarded from the cache to make space for other documents.
- Updates: Misses occurring when accessing documents previously requested but already expired.
- Uncacheable: Misses occurring when accessing documents that need to be delivered from the final server (e.g. dynamic documents generated from cgi-bin scripts or fast changing documents)

[*]

[*] The Introduction is in large parts a copy of the paper "Large Caches and Satellite Distribution" of Pablo Rodriguez and Ernst W. Biersack []

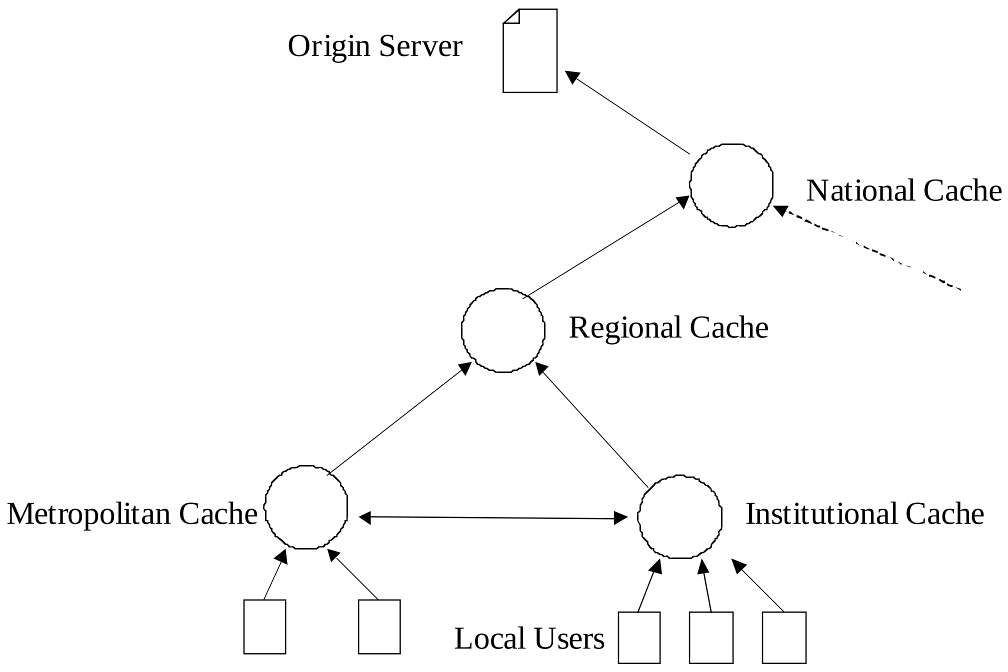

Figure 1 – Hierarchical caching

In this report we will focus on the upper boundary which hit rates might achieve. More detailed information can be found in [2]. The more users share the cache, the greater the chance that several clients are interested in the same document. One popular scheme to make caches cooperate is via a caching hierarchy (Figure 1). In a caching hierarchy caches are placed at several levels of the network, including caches at the client side, at the metropolitan networks, at the regional networks, and at the national backbones. Caches cooperate at the same level of the caching hierarchy and at different levels of the caching hierarchy, sharing all the documents requested by their clients.

3 Choosing parent & siblings for Eurecom Webcache

To do webcaching a proxy-cache needs to be installed. We use Squid 1.1.22 [1] which is a wide spread high-performance proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects. Squid is Free Software and copyrighted by the University of California San Diego and Duane Wessels.
.

3.1. Parent & sibling caches

One of the first things to do was finding from the cache list [3] the "nearest" and "fastest" caches. But this terms are not sufficient to classify a cache. One cache may have a fast response but on the other hand it has a low throughput. Another cache may respond slower but have a high throughput. The result of a first search gave us the following caches in our region:

cache.unice.fr
cache.imt-mrs.fr
cache.cma.fr
zig.inria.fr
schubert.obs-nice.fr
www.univ-aix.fr
www.univ-aix.fr
ache.sophia.cnrs.fr
www.ceram.fr
iutsoph.unice.fr
proxy.alcyonis.fr
coriolis.sophia.cnrs.fr

From this list wee took the following caches after contacting the system administrators and added them as sibling:

cache.unice.fr = 134.59.1.31
coriolis.sophia.cnrs.fr = 195.220.194.194
proxy.alcyonis.fr = 195.78.3.6
iutsoph.unice.fr = 134.59.136.5

Because of some problems we had with the www.unice.fr cache **-** Eurecom people started to complain about not updated documents (expired documents & sometimes down) **-** its status has been changed from parent to sibling. This allows us to be more independent and flexible. Our decision was based on the availability and the measured ping time as illustrated in Figure 2. In fact it is not even necessary to measure the ping round trip time because the averages are approximately proportional to the number of hops. For these measurements we use ping with packet size 64 bytes.

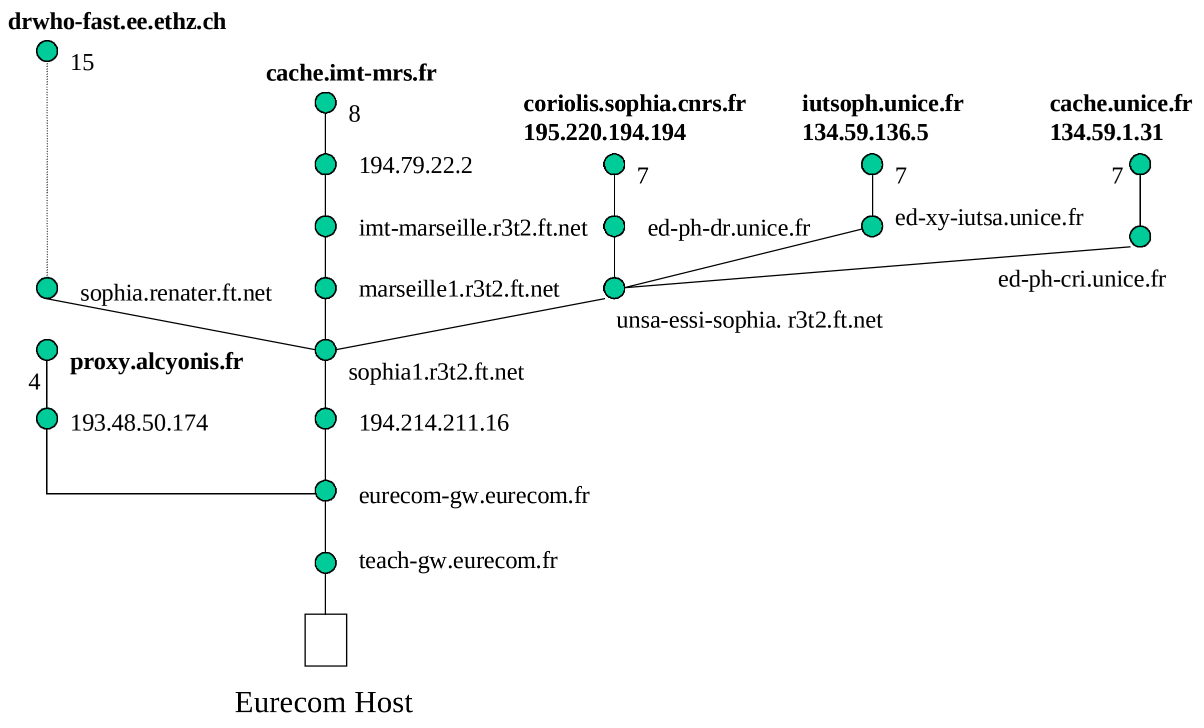

Figure 2 – Physical connection to the caches

However ping time are not the only parameter characterizing a cache response time more sophisticated methods are shown in the next chapter.

4 Non-cacheable objects

4.1. Squid configuration for non-cacheable objects

In the World Wide Web there are several documents which can't be cached such as cookies, CGI scripts (they generate a user specific HTML output), secure HTTP etc. Squid will take into account these words using the following lines in the configuration file “squid.conf”.

```
hierarchy_stoplist cgi-bin https Set-Cookie
```

Squid will not query neighbor caches for URLs containing these words. Thus these objects will be handled directly by this cache.

```
cache_stoplist cgi-bin ? eurecom.fr https
```

A list of words which, if found in a URL, cause the object to immediately removed from the cache. In other words, use this to force certain objects to never be cached. You may list this option multiple times.

4.2. Other non-cacheable objects

In addition to the objects already mentioned Squid configuration for non-cacheable objectsabove the HTTP protocol allows to control cache behavior in the MIME header. Objects which contains one or more of the following entries do not get cached:

- Stop list as mentioned above.
- Cache-Control: no-cache
- Cache-Control: no-store
- Pragma: no-cache
- Pragma: max-age=0
- Expires: 0
- Authorization
- All HTTP methods other than GET

In a MIME Header it is possible that several entries mentioned above occur at the same time. We searched all entries for headers with two reasons of non-cacheability. We found that only two combinations occur very often: “Cache-Control: no-cache” combined with “ Pragma: max-age=0” and “Cache-Control: no-cache” combined with Pragma: no-cache”.

4.3. Evaluation of non cacheable objects

In the figure below we can see the amount of non cacheable objects for some days at Eurecom’s webcache. The figure is a result from the Perl script [4] “evalCache.p” which generates extensive statistics of a squid access log file. It is easy to see that the percentage of non cacheable objects does not exceed 10% and is a minor part of the traffic these days. In future things may change because dynamically and fully personalized generated web pages become more and more popular.

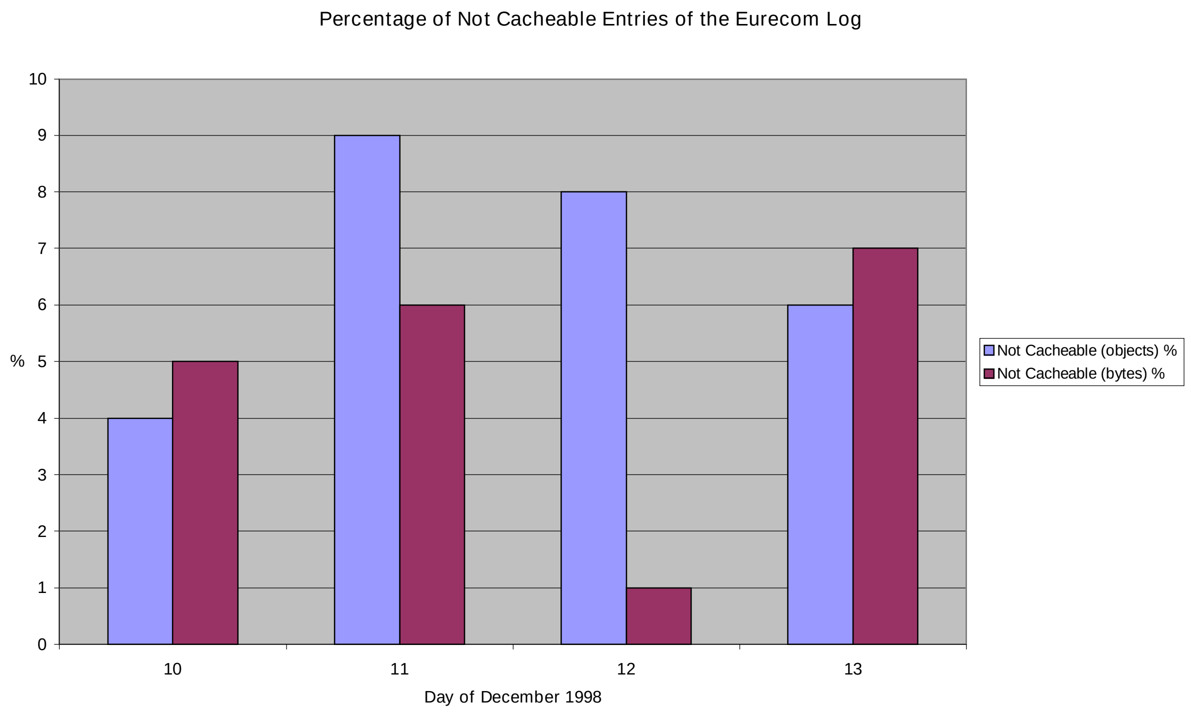

Figure 3 – Percentage of not cacheable entries of the Eurecom log file [*]

5 Performance testing

5.1. Extensive Squid access log statistics

We wrote a Perl script “evalCache.p” which takes the squid access log file as input and dynamically generates a web page with extensive log statistics [5]. An example print out is given in the Annex. From this data source we obtained the following distribution of Eureocm log entries in the period from December 10 to December 13.

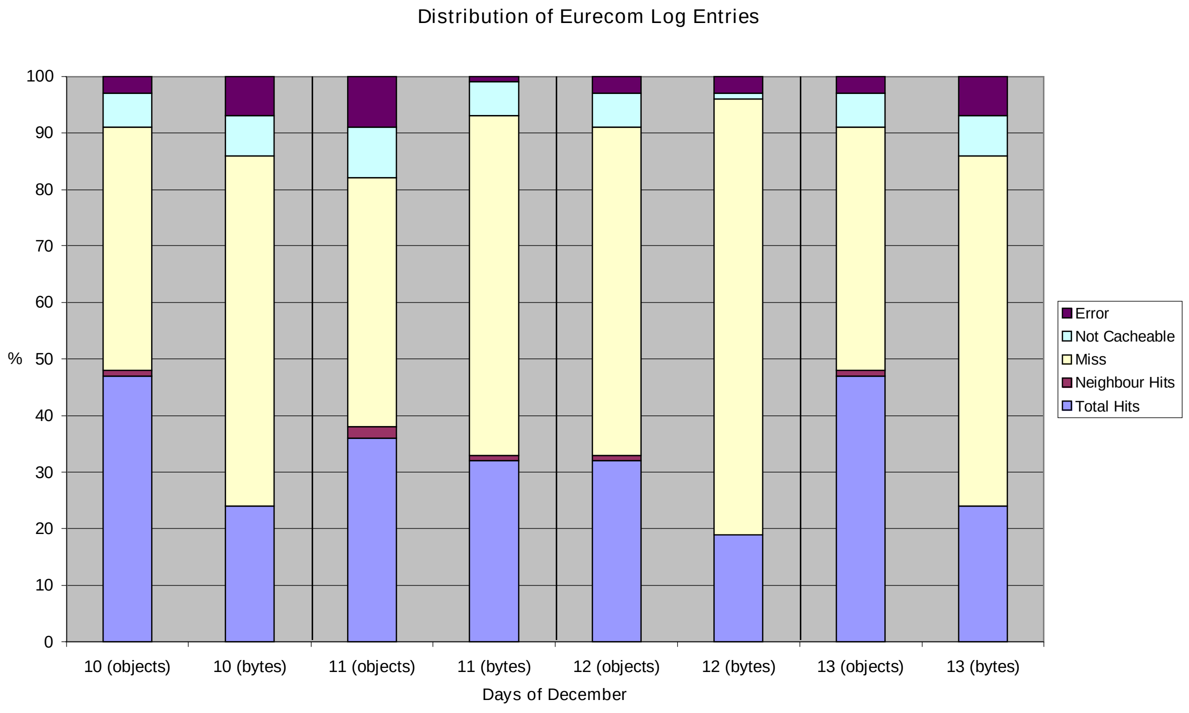

Figure 4 – General distribution of Eurecom Log Entries

For further explanation of the legend we refer to the Annex Perl script output for cache analysisor directly on the Web page “Description of all the expressions used in the homepage” [5]. If we look at the byte rate the variation is low. We see that there are few sibling hits. This might be due to small and extremely heterogeneous client population. The error rate as well as the not cacheable rate is small and does not exceed 10% each. All the traffic of ICP (ICP is the inter cache protocol to find out if a document is cached in a neighbor/parent cache) is transported on UDP packets and are not regarded throughout our work. Each neighbor hit precedes a UDP hit. Only very small objects are directly included in the UDP packet. These hits result in a special TCP log entry which we consider as an error. Their occurrence is negligible.

[*] For all charts with % as Y-axes title, means percentage of all log entries

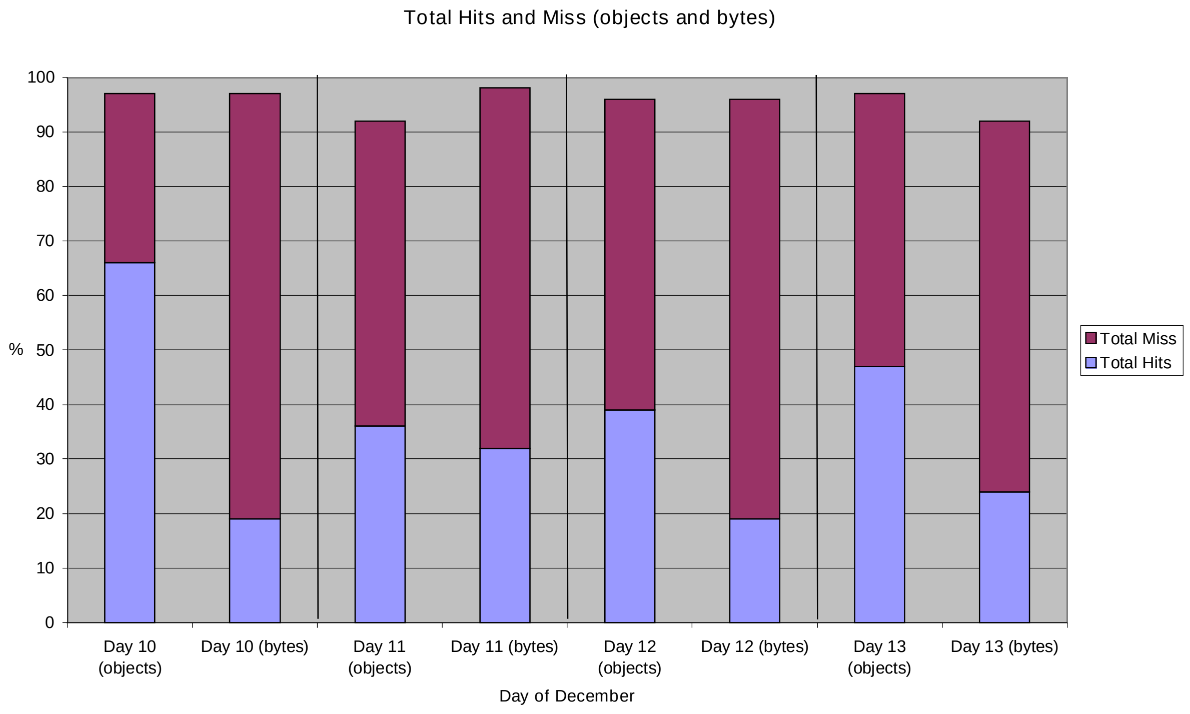

Figure 5 – Total Hit and Miss of Eurecom Log Entries

Here we distinguish only Hits and Miss. Not cacheable objects, misses and neighbour hits are counted as a miss. This is to better evaluate our own cache without considering neighbors. The hit byte rate never reaches 40% of the measured traffic.

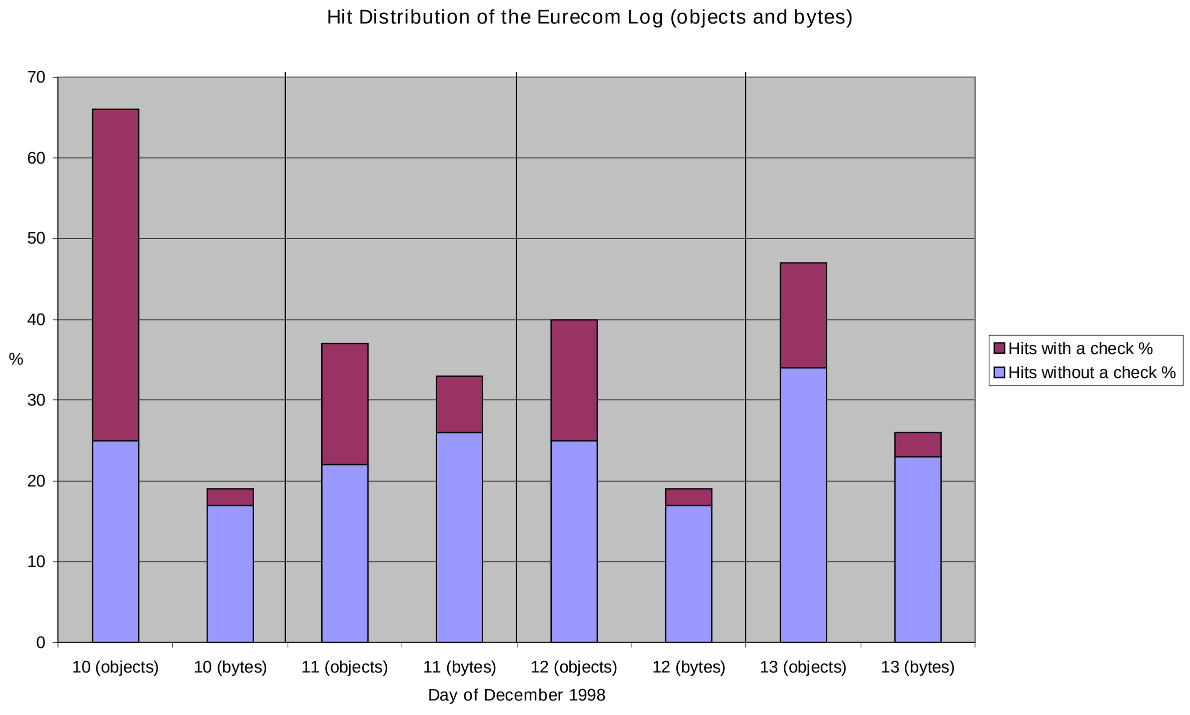

Figure 6 – Hit Distribution of the Eurecom cache

One problem of web caching is that an object that is in the cache might change its content at the origin server. When a client requests such object from the cache, it returns the object from the memory, but this object is obsolete. To avoid this behavior several measures are taken:

- The cache heuristic if an object is stale or not
- An If-Modified-Since request is sent
- The originate server set the "expires" header (supported by HTTP request but not used nowadays)

A cache can estimate if an object is valid or not using a simple heuristic: It looks at the time LMS (Last-Modified-Since) which is always delivered by the originate server (the HTTP protocol allows all active network devices to discard objects without this field). The estimation is done as follows: If an object was valid from the LMS time to the time it was requested, it will still be valid later for the period of 30% of the interval between these time stamps. This is called the heuristic of the cache.
If the second request for the same document exceeds the time calculated with the heuristic the cache checks with a IMS(If-Modified-Since) request if the document has been changed. If yes, the document is downloaded again and updated in the cache. If no, the document in the cache can immediately been delivered to the client and the heuristic gets recalculated.
We see in Figure 6 that the major part of the request satisfy this heuristic. The "Expires" would override the heuristic but unfortunately "Expires" is not used nowadays. Additional graphics may be found in the Annex.

A client has also the possibility to force an IMS request and override the heuristic by pressing the reload button in Netscape™ for example. We see in Figure 7 that in about the half of these requests the heuristics would have resulted in the delivery of an obsolete object. In fact a user normally assumes that a document has changed because of the information he got from the old document for example. If a user visits a news page where information is from yesterday he will certainly press the reload button.

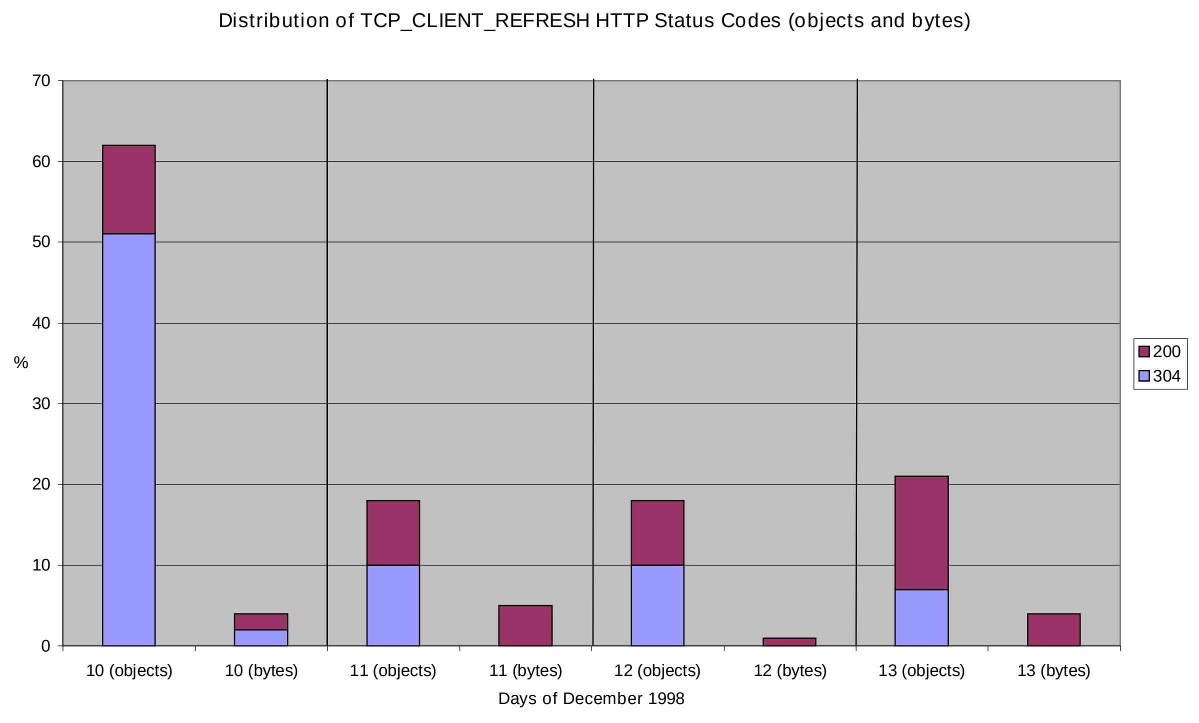

Figure 7 – Distribution of TCP_CLIENT_REFRESH in the Eurecom Log

In the Squid log file we can not distinguish between a "First Access Miss" and a "Capacity Miss". But the sum of these two are the subtraction of non cacheable objects of all misses. We see in Figure 8 that there is a large potential to increase the hit rate at Eurecom cache by increasing the cache memory to decrease the capacity misses. Another possibility would be prefetching as we will discuss below.

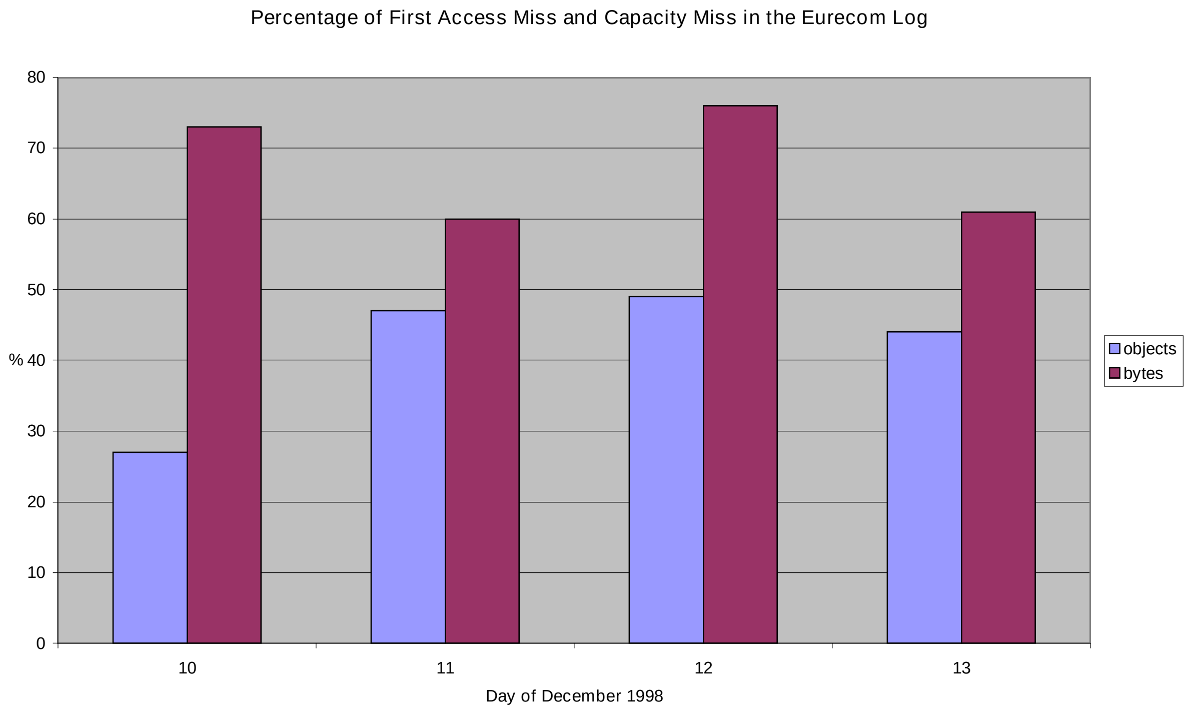

Figure 8 – Percentage of First Access Miss and Capacity Miss in the Eurecom Log

6 Layer 2 & Layer 7 RTT measurement

6.1. Introduction to the developed programs and scripts for layer 7 round trip time measurement

To decide which are the best neighbor caches we wanted to measure the end-to-end round trip time. In fact this gives some problems:

- Usually an external HTTP client is not allowed to access a cache from outside
- The time between measurements from different caches should be as small as possible
- Different document sizes should be tested at the same time

To realize these facts we wrote a c-program (see Annex RTT-estimation program "cache.c") which forks for all given caches and all documents a child process. These child processes then will send the HTTP request to the corresponding server and measure the time it takes until all bytes are received. The parent process collects these data and writes them into the log-file. Afterwards data may be treated by any other script. The program needs the following input files:

- a file containing all URLs
- a file containing all caches

All these files are text files and easy to create. In the following table is a small description of the line format of these configuration files.

File	***Format***
URLs	`http-server http-request`
caches	`cache-name port-number`

Table 1 – Input files for RTT-estimation program

In the "caches" file the word "**`SOURCE`**" is a string with special meaning and indicates to send the request directly to the source without passing any cache.

The output file "cache.dat" is of the form:

Output File Format
`Unix-time-stamp seconds microseconds size cache-address http-request`

Table 2 – Output file format for RTT-estimation program

At the beginning of each measurement block a special line containing a timestamp is introduced. This will enable easier treatment in a Perl-script afterwards. All measurements are averages of up to three samples. In addition we drop the first measurement every time, because it may be that the object is not in the tested cache.

This program is invoked by a small scheduler which launches the process every 15 minutes. At the same time genuine ping measurements were taken and logged in file "ping.dat".

6.2. Results from the end-to-end RTT and Ping measurements

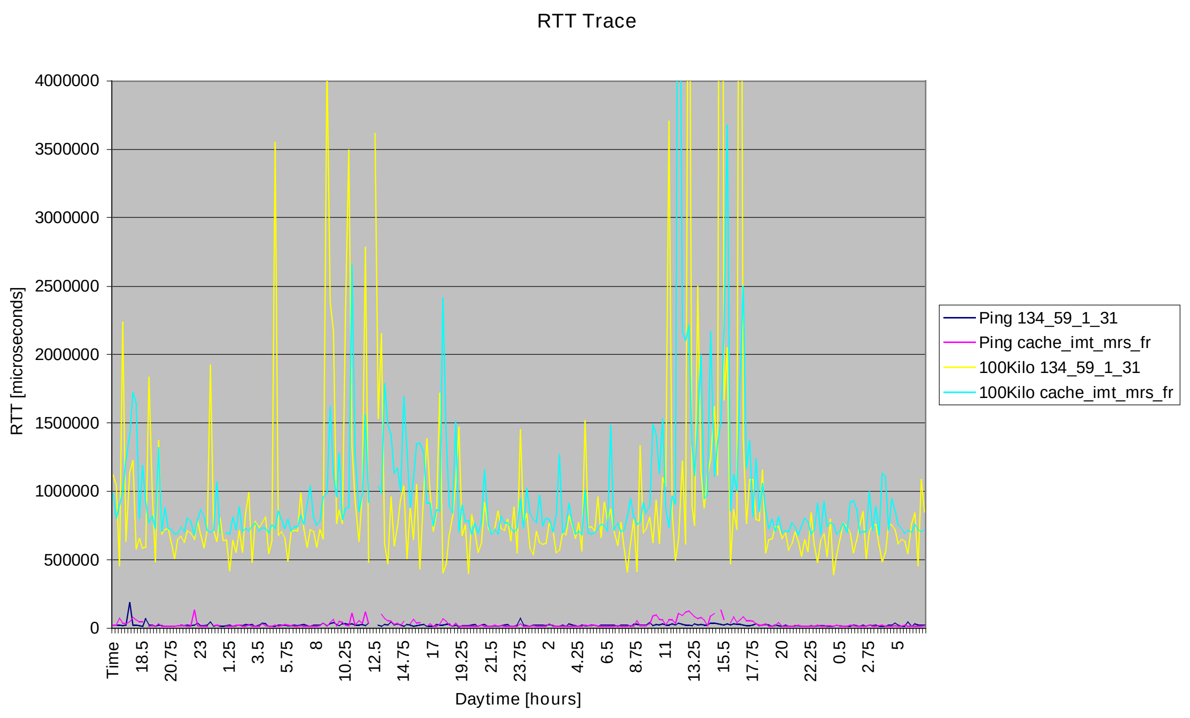

Figure 9 – End-to-End RTT and Ping measurements

We see here the RTT measurements done over 3 days (8. – 11. December 1998). When we talk about layer 2 measurements, we mean the genuine ping measurements. Layer 7 measurements are measurements taken with the program described in Introduction to the developed programs and scripts for layer 7 round trip time measurement. This measurements include the time from the start of the request, passing through a cache until the document is completely downloaded. The figure shows pretty well the increase of traffic during the work hours and especially in the afternoon. More detailed graphics may be found in the Annex.

In the above and the following figures and calculations we cut measurements samples larger than 3 times the standard deviation of the whole sample population to reduce the risk of misleading results caused by single outlying measurements.

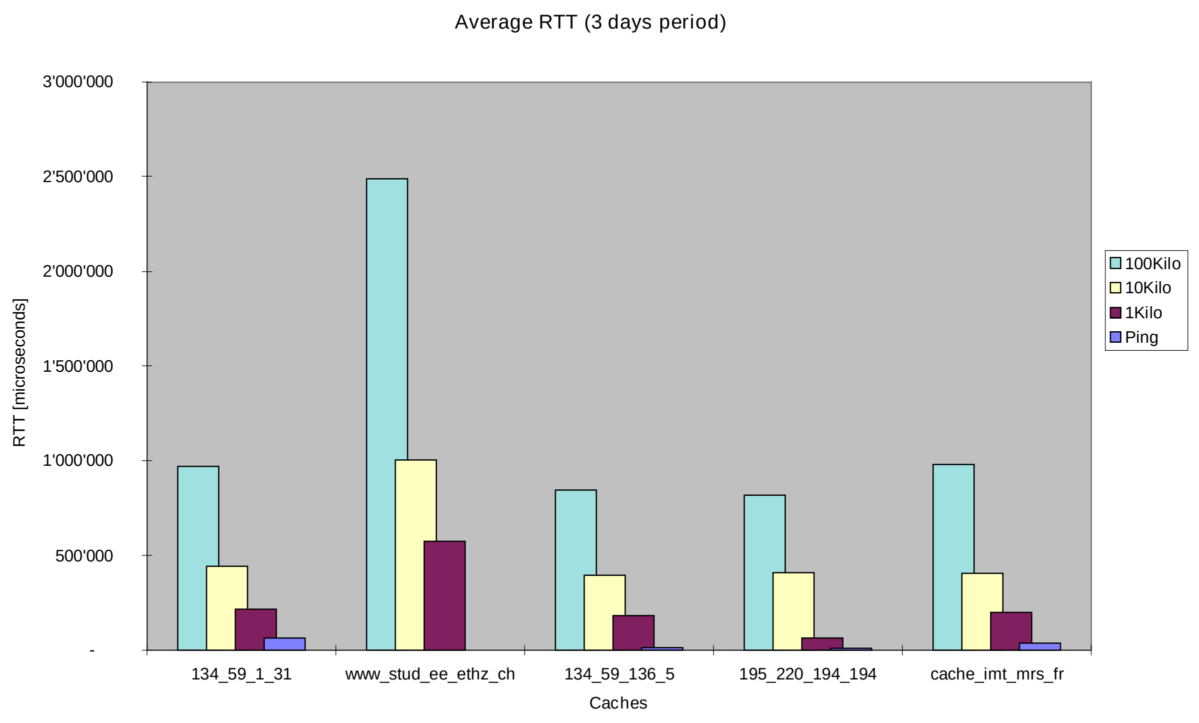

Figure 10 – Average RTT

We see that every cache responds faster than accessing directly to the origin server in our case at ETH Zurich where the three files with different sizes (1kB, 10kB and 100kB) are stored. This is a very important result because the end user sees less latency and no significant traffic on the internet is generated.

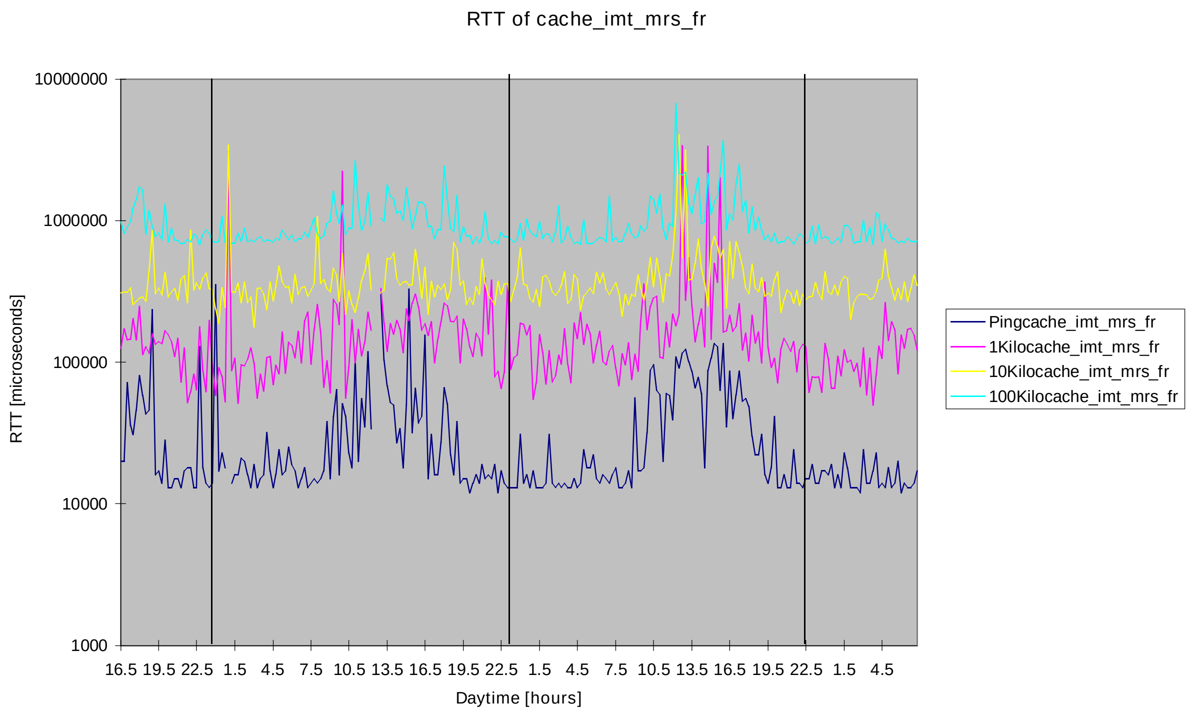

Figure 11 – RTT of cache.imt-mrs.fr

If we take a look at the RTT for 1kB and 100kB they differ 2 decades in size but only one in the round trip time. Overall we can say that large documents have a higher throughput than small ones.

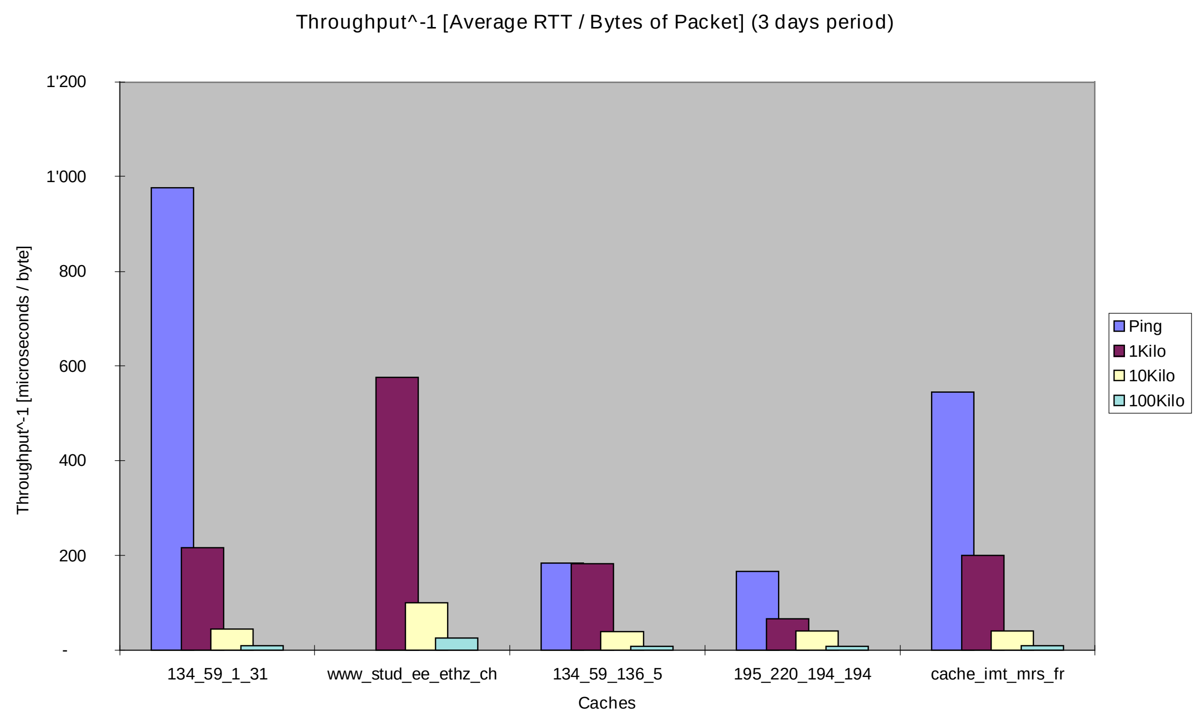

Figure 12 – Average RTT over Bytes of Packet

The fact mentioned above is true for all caches we examined.

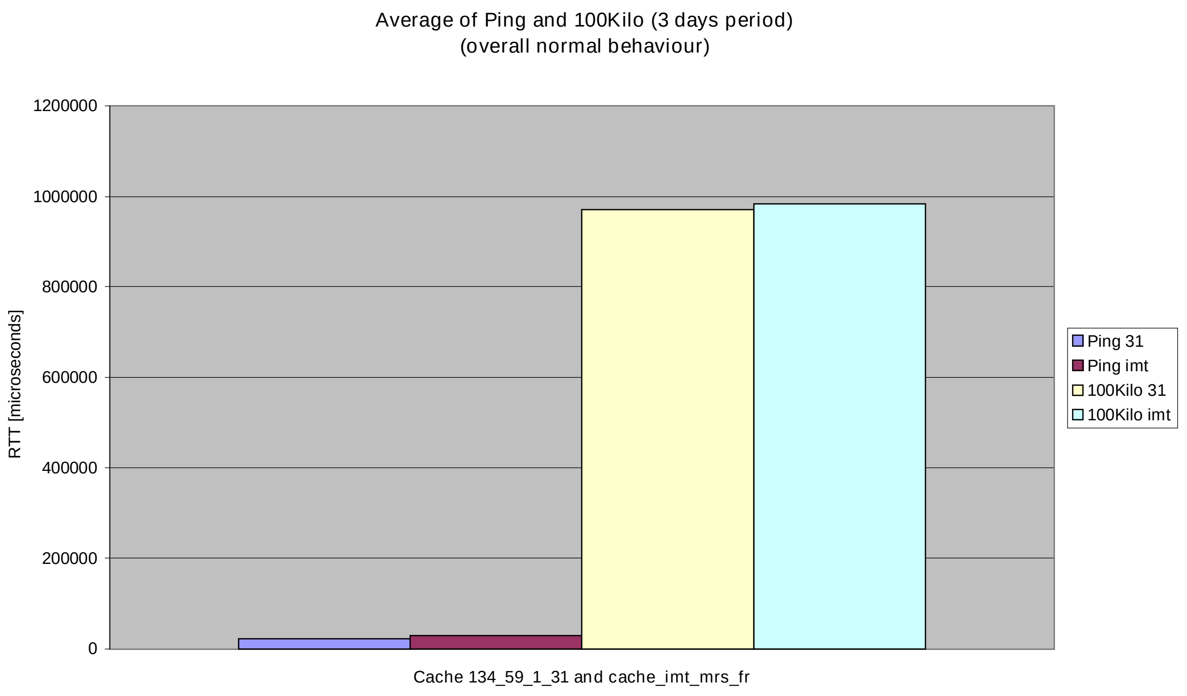

Figure 13 – Average RTT over Bytes of Packet

To choose the best cache available at the moment we could use the ping measurements as a heuristic. Is this reasonable? As we can see in the averages over three days we will do the right choice following the ping measurements. In fact it seems to be a good heuristic.
If wee have a closer look on the trace we can observe small intervals where this is not true. During this intervals we would do a bad choice following the ping measurements. The next chart shows such an interval of one hour. To search for these intervals we subtract the ping measurements of one cache from those of another one. We do the same for the

100kB file measurements. Now we just have to find a period where the sign of these two curves is reversed (one curve in the positive area and the other curve in the negative area). The average over this small periods shows also this inversion. We would like to recall that all samples are averages of several single measurements.

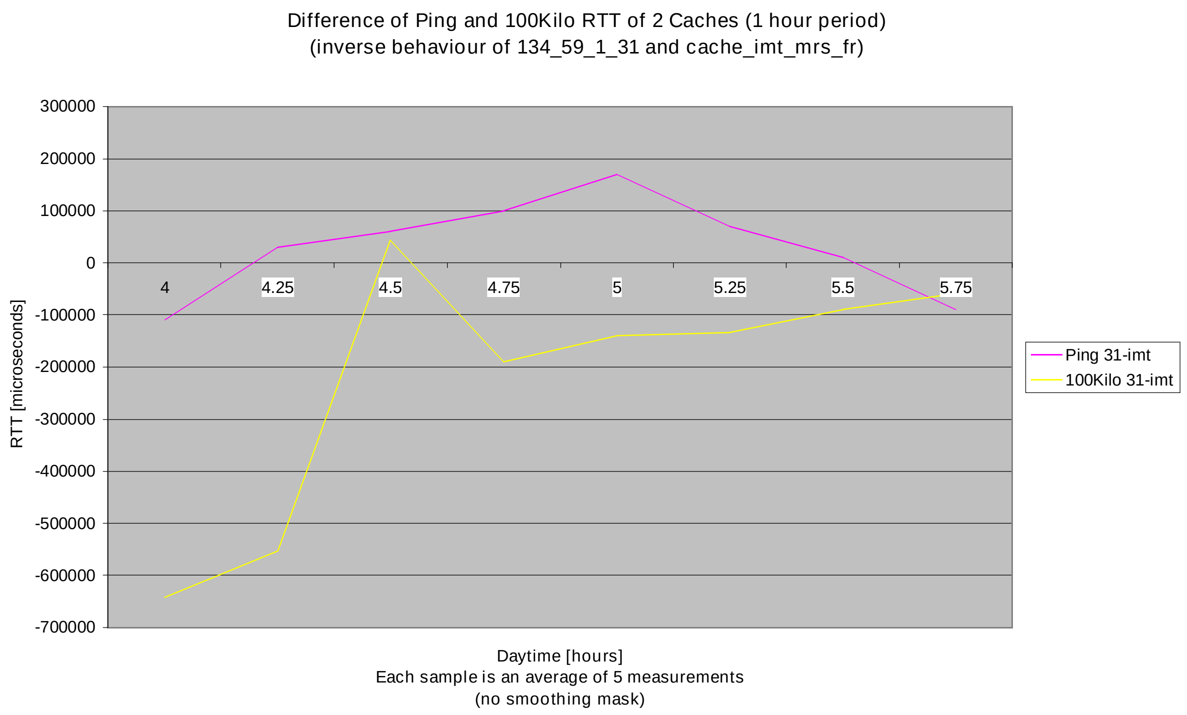

Figure 14 – Inverse behavior

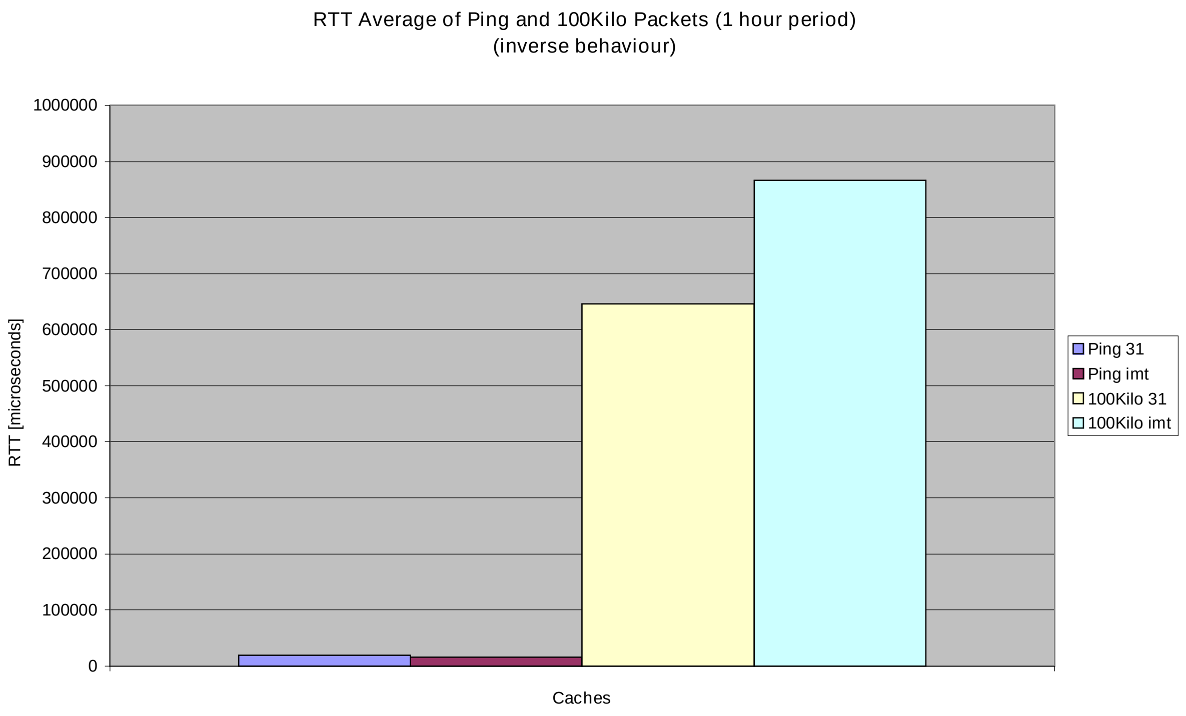

Figure 15 Inverse behavior, Average

In the Annex you can find a series of charts using a smoothing mask with different smoothing periods. For each measurement we average a certain amount of the following measurements. The larger the smoothing window is, the more we find significant inversions. But if we take a closer look to the samples without averaging there is hardly any inversion.

7 Satellite Broadcast Project

7.1. Introduction

Traditional hierarchical webcaching is performed through the internet. Now we can imagine a broadcast based cache-updating on a separate network using satellites [6]. Cacheable objects are pushed via satellite to the local edge caches of the internet without passing the internet. This will decrease the load on the internet and increase the hitrate in local caches. Of course we need enough disk space to save these documents. In fact, using intelligent agents which track local user behavior, only a part of the distributed objects may be kept in local memory and leed to better hitrates. This is purpose of actual and future research.

7.2. Maximal possible hitrate

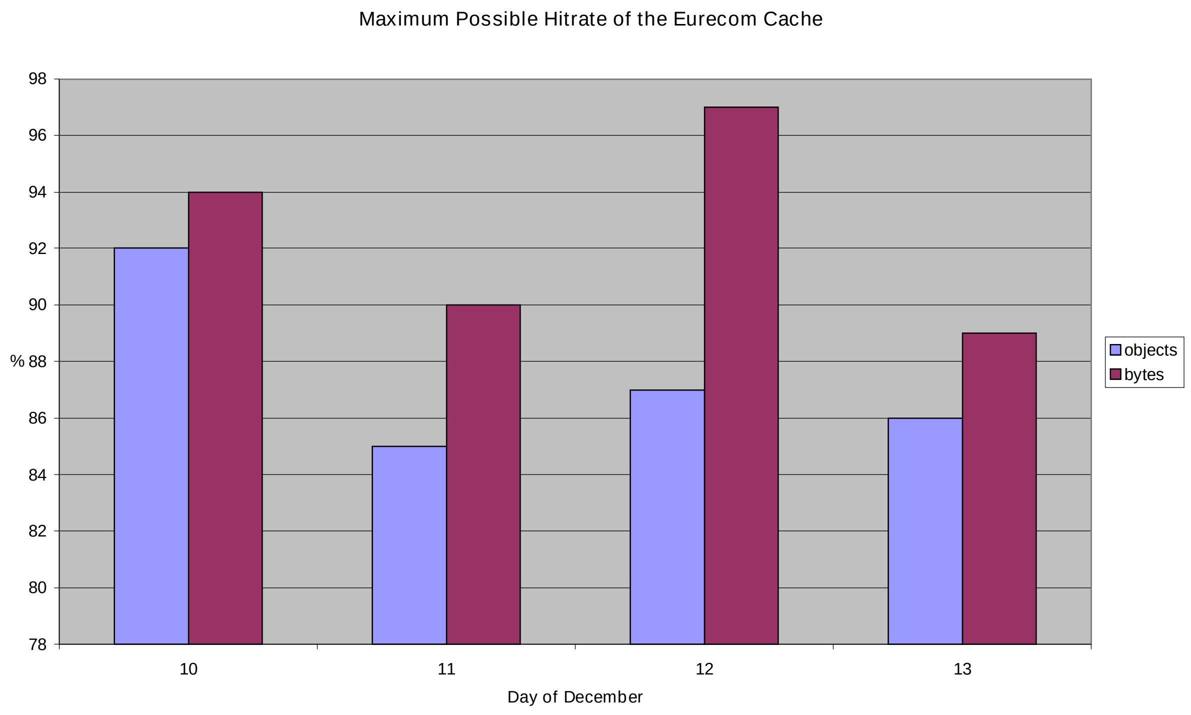

Figure 16 – Maximum possible Hit rate of Eurecom Cache

To calculate the maximum possible hit rate we imagine having an infinite cache memory size. The joke is now, using the satellite cache broadcasting for cacheable documents we will have all cacheable objects already in our cache before Mr X asks for it. What will be the new hit rate we could achieve? We obtain this value by summing all total hits (TH), first access misses (FM), capacity misses (CA) and TCP_REFERESH_MISSes (RM). The result for Eurecom is given in Figure 16. In the average we could achieve a hitrate of over 90%!

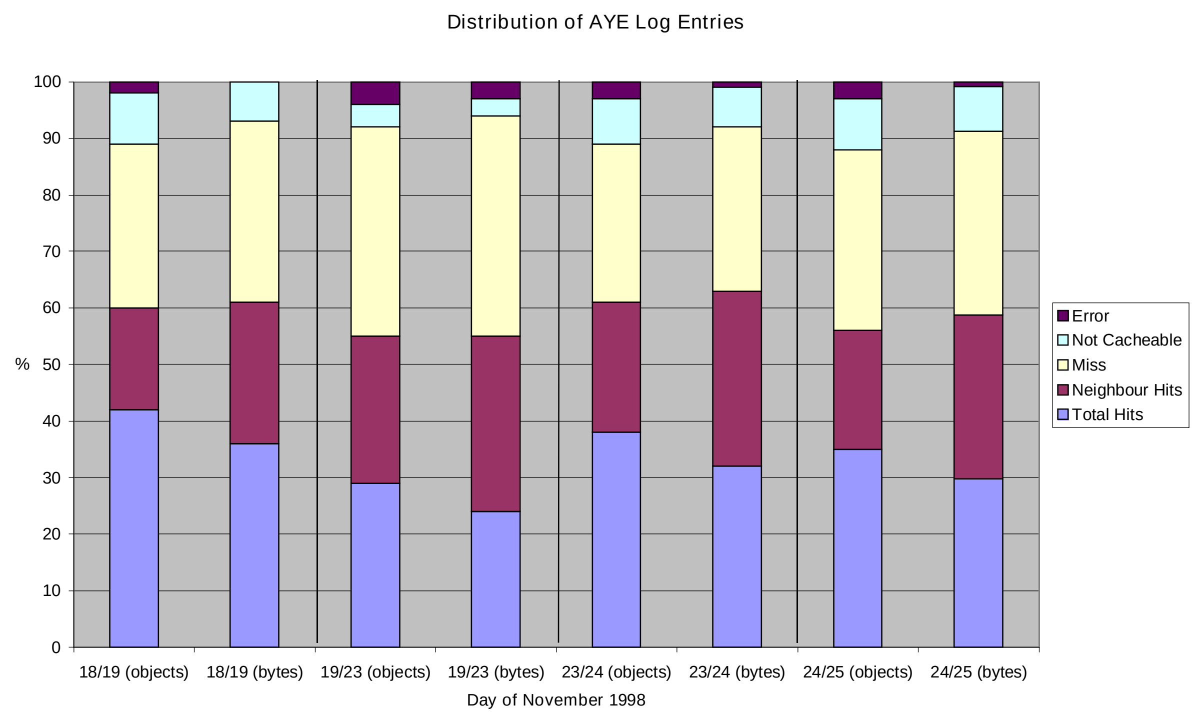

Figure 17 – Distribution of AYE Log Entries

In this distribution, compared to the previous one from Eurecom we see have a similar hit rate and not cacheable rate, but a nice amount of neighbor hits. We think this comes from the large population using AYE cache.

To know if it is worth installing satellite broadcasts we compared access logs from two large proxy caches in the U.S.: AYE and NLANR. For every miss in AYE cache we searched for a possible entry in the NLANR cache. The encountered hit-candidates need then to be checked precisely:

- The logged line in the satellite cache must before the local request
- Compare LMT (last modified time) if it exists
- Compare expiration date if it exists
- Compare file size

The flowchart in the Annex explains in detail how to perform the tests. the Perl-script "sat.pl" realizes this approach. Unfortunately this search is very slow and we estimated the calcution to take 10 years on a Sun Ultra 10. That is why it can be used only for small access log files. Besides it is an interesting question how much we could decrease the traffic due to IMS-polling. The very similar flowchart is in the Annex too.

Our second approach consisted in using Glimpse, a fast indexing and query system that allows you to search through a large set of files very quickly [7].

Before we can use Glimpse the data file has to be indexed with "`glimpseindex`".
Glimpse doesn't allow search for strings larger than 32 characters. That's why we filter the local cache data as much as possible and store only the URL and the document size.

What to filter	***Local logfile***	***Remote logfile***
TCP_MISS	X	
TCP_REFRESH_MISS	X	
TCP_CLIENT_REFRESH_200 X		
Cut UDP entries		X
Cacheable	X	X
Extract URL	X	X
Cut "http://"	X	X
Exchange Special Characters	X	
Extract last 32 characters of URL	X	
Check Special Character at the beginning	X	

Table 3 – Filter criteria

Using this filtering and glimpse for fast search the calculation time will be reduced to about 10 days.

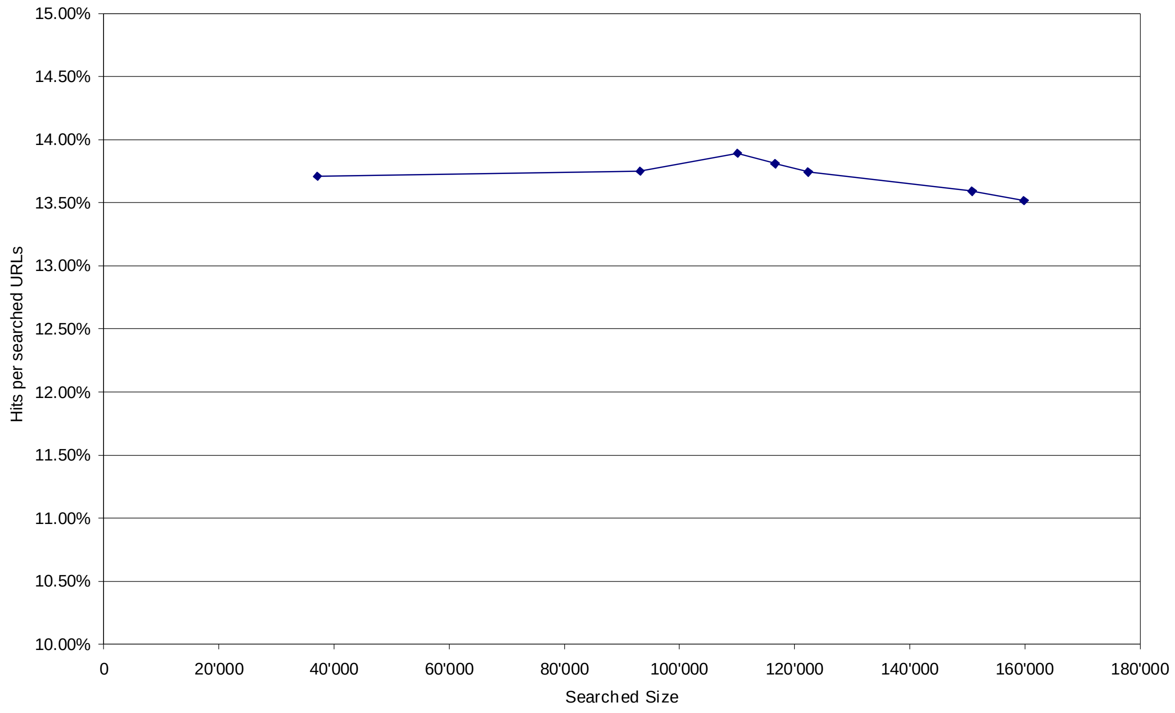

Figure 18 – Increase of the hitrate when using NLANR as "satellite source" for AYE cache

In Figure 18 the variation of the mean hit rate found for different progress of the search is shown. The X-axis represents the number of hits found per line. The number of lines searched in is the Y-axis. As expected for exponential distributed expiring and Poisson distributed average request rate, the graphic shows, that the hit rate is almost constant in function of the position of the search. This allows us to give a result for the approximate value we will obtain when searched through all URLs without completing the search. The mean value is about 13.7 %.

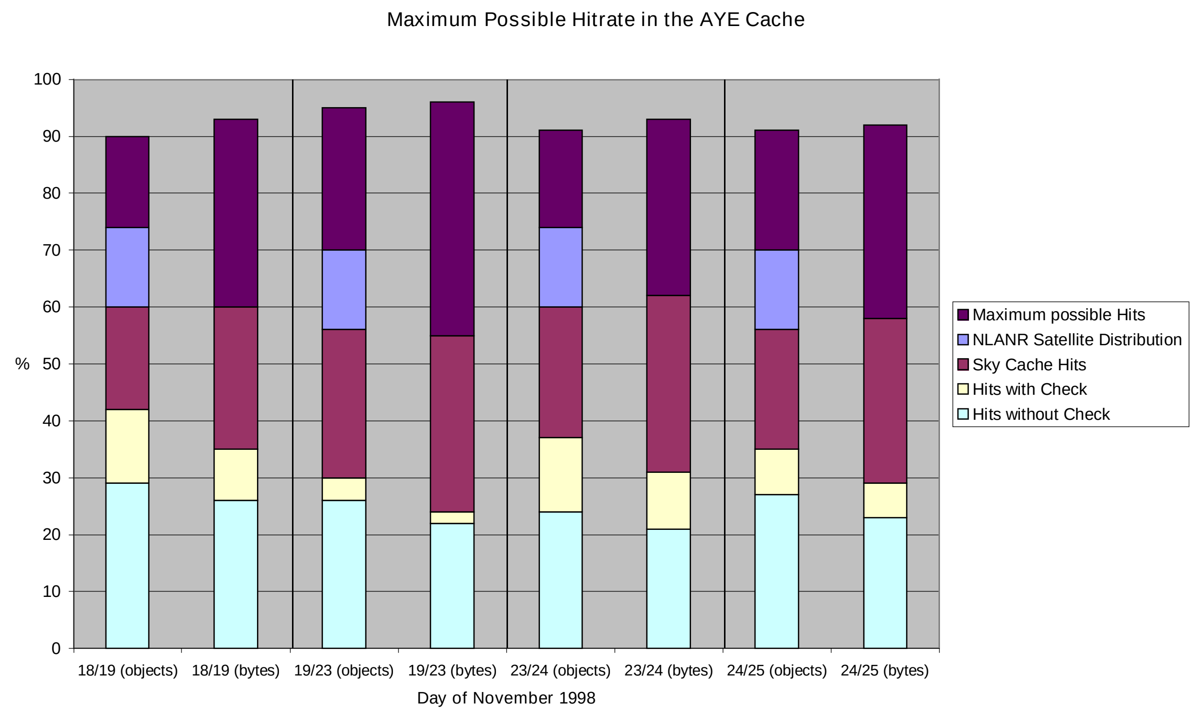

Figure 19 – Maximum Possible Hit rate in the AYE Cache

The Figure 19 shows us how close we could get with the Skycache and NLANR cache to the maximum possible hit rate. The contribution of the NLANR is clearly the upper bound for the hit rate. We can not say until now how much of the 13.7% would be left in a real satellite implementation. Further and more detailed examination of the log files is necessary.

8 Conlusion

For a large population webcaching is a very efficient tool to decrease internet traffic and latency. However caches have a limited performance due to the high number of objects not hit in the cache. In future there will be more and more caches communicating more and more efficiently between eachother. For example the bandwith nowadays used for ICP traffic will be replaced by updates of cache content listings eventually through special channels.

9 Acknowledgments

We would like to thank Pablo Rodriguez for his help and support in our project.
The research in webcaching is depends on good relation with neighbor caches where we would like to thank those system administrators who complied with our wishes.

10 Annex: Scripts & Programs realized

10.1.RTT-estimation program “cache.c”

10.2.RTT Scheduler “run.c”

10.3.Perl script output for cache analysis

- Hit and Miss extensive Analysis
- More Details
- Description of all the Expressions

10.4.Satellite distribution: Perl script “sat.pl” to calculate increase of the hitrate comparing “Expires”, “LMT Last modified” and size using a given heuristic and flowcharts

10.5.Satellite distribution: Perl scripts and Shell script to find the increase of the hitrate by comparing only the URLs and using Glimpse

10.6.Graphics

11 References

1[]	http://squid.nlanr.net
2[]	Pablo Rodriguez and Ernst W. Biersack, “Large Caches and Satellite Distribution", Eurecom 1998
3[]	http://cache.unice.fr
http://www.serveurs-nationaux.jussieu.fr/cache/
4[]	Larry Wall & Randal L. Schwartz, “Programming Perl”, 1991
5[]	http://lantana.eurecom.fr:8080/~webcache/stat.html
6[]	http://www.skycache.com
7[]	http://glimpse.cs.arizona.edu