



TERENA Technical Report

Seafood: Extended Cache Statistics

Jens-S. Vöckler voeckler@rvs.uni-hannover.de

Institute for Computer Networks and Distributed Systems (RVS)
University of Hanover
Germany

April 2000

Summary

Seafood is the name of a software suite for analysis of data extracted from web cache log files. Log files using the wide-spread Squid-style native format are processed offline and stored in an SQL database. A web-based interface is used to access typical selections as part of Seafood. The long-term and easy-to-access storage allows for trend analysis as well as complex views and correlations between different data sets and members of a cache mesh.

The software is available from:

<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/>

For further information please contact:

TERENA Secretariat
Singel 468 D
1017 AW Amsterdam
The Netherlands

Tel: +31 20 530 4488
Fax: +31 20 530 4499
E-mail: secretariat@terena.nl
WWW : <http://www.terena.nl/>

© TERENA 2000, all rights reserved

Parts of this report may be freely copied unaltered, provided that the original source is acknowledged and the copyright preserved.

Contents

1	Introduction	4
1.1	Proposal	4
1.2	Objectives	5
1.3	Deliverables	5
1.4	Acknowledgements	5
2	Specification	6
2.1	Introductory Comments	6
2.2	Taxonomy	7
2.3	Mandatory Sections	8
2.4	Extensions	10
3	Implementation	11
3.1	Compilation and Installation	11
3.2	Configuration	12
3.3	Running Seafood	15
3.4	The Textual Results	17
3.5	Parsing a URL	18
3.6	Selected Internals	19
4	Database Design	20
4.1	Database Implementation	20
4.2	Filling the Database	24
5	Web Interface	25
5.1	The Selection Menu	25
5.2	Implementation of the Prototypical Interface	29
5.3	Example for an Implementation of the SQL Queries	30
6	Conclusions	32
7	References	32
8	Further Information	34
	Appendix A Software	34
	Appendix B Deviations from the Specification	34
	Appendix C Implementation Details	39

CONTENTS

1 Introduction

Seafood is a software suite for offline analysis of access.log files from the widespread Squid web caches [6] and similar formats. The resulting summary can be viewed as a text file containing tables in the tradition of Calamaris [8]. Another output option gathers data extracts into an SQL database in order to create a long-term, easy-to-access store. The necessary scripts for collecting summaries and retrieving graphical results via a web user interface are part of the Seafood software suite. Using an SQL database allows for trend analysis, complex views and correlations between different data sets as well as providing a source for data mining.

1.1 Proposal

A large number of administrators in today's caching community maintain cache servers based on the Squid software [6]. The development of the Squid software during the last year has resulted in new versions with an amazing improvement of performance, flexibility and stability. In contrast to the development of the caching software itself, there is a lack of accompanying tools to administer a Squid cache and to measure its effectiveness. The purpose of this project was to develop tools to gather and extract statistics from a number of Squid caches and present the results in a visual way.

The Squid software generates log files that enable the administrator to find out interesting information about the caching service - if the necessary numbers can be successfully extracted from the sheer amount of data. Due to the size of the log files, and for performance issues in generating long term statistics, many cache maintainers only look into these log files occasionally to get an impression of the current situation.

The log file is usually processed offline, which offers a number of advantages. Online processing would require constant polling of the data of interest from the Squid cache whilst the cache was operating. Offline processing allows the administrator to arbitrarily select the time period and the detail of the output whilst processing the file.

Still, most solutions suffer from the volume of possible and interesting data. Also, most data gets more interesting, if combined with other related data. Thus the log file parsing results must be easy to incorporate into an SQL database. From this database, different sets of simple data can be put together into a more complex view of the behaviour of a cache or set of caches, all with simple SQL statements. However, most users, and even administrators, prefer easy-to-handle tools. Thus a custom-tailored web interface, based on the database, should be able to produce graphs on-the-fly, based on the data sets selected by the user.

Currently, there are a few log file processors available, but all suffer from limitations. Some processors will give numbers, some will generate long ASCII based reports, whilst others are designed to produce images with coloured graphs. Additionally, there is no Squid statistic tool that supports an interface to standard databases.

A very well known processor is 'Calamaris' [8], which is implemented in Perl. Each weekday the 10 caches in the DFN caching service accumulate well over 4 GBs of log file data. Calamaris spends over 18 hours on a high-performance workstation in order to process this data. Other log file processors might be faster, but less detailed in their output.

For performance reasons and to get an impression on Calamaris' potential, a prototype implementation of Calamaris was developed in C++. This port is sufficiently faster, and might even be sped up further for multiprocessor and/or multihost environments, but currently lacks support for the up to date version 2.x of Squid.

1.2 Objectives

A promising prototype approach of the well-known Calamaris tool to the better performing C++ is already done, as far as Squid 1.x log files are concerned. So far, this prototype only produces textual results. The proposed project is to look into parsing Squid 2.x and (perhaps) NetCache [7] results, and return the results in a way that is easy to incorporate into a database. Furthermore, a prototype web interface module is to be prepared which shows how different on-the-fly views of data can be achieved.

1.3 Deliverables

The scope of the work consists of several issues:

- In co-operation with the TF-CACHE group, acquire a set of interesting, yet simple, information to be extracted from the log files, preferably based on the information already available in Calamaris.
- Port the C++ version to parse Squid-2 log files and preferably also NetCache log files. The results from the previous phase should be incorporated into this phase.
- Design an efficient scheme to store results in a relational database. Linear growth of the database might or might not be desirable; the topic will have to be discussed with TF-CACHE. If linear growth is not desirable, a scheme to reduce older data will have to be developed.
- Implementation of a prototype web interface to extract interesting views from the database and generate on-the-fly graphs to present the results.

1.4 Acknowledgements

The TERENA project "Extended Cache Statistics" was accepted on 4 December 1998, and ran from 1 April 1999 till 31 December 1999. The project was conducted by staff working for the DFN Caching project at the Institute for Computer Networks and Distributed Systems [1] of the University of Hanover, Germany. The DFN caching project is sponsored by the German Ministry of Education, Science, Research and Technology (BMBF - [2]) through the German Research Network Association (DFN Verein - [3]). The Trans-European Research and Education Networking Association

(TERENA - [4]) sponsored the "Extended Cache Statistics" project, which was aimed at the exploration of data by correlation through a database. A side product is the long-termed availability of data. The project was proposed by TF-CACHE, the TERENA Caching task force [5].

2 Specification

The project "Extended Cache Statistics" focuses on delivering a long term view of data obtained from cache log files originating from cache servers running Squid or possibly NetCache software. The purpose of the specification is to outline a mandatory set of data to retrieve from the log files. Note that problems related to the extraction of the data are not addressed at this point. The database design will also be addressed later.

2.1 Introductory Comments

With the help of a SQL database, different views to the various performance data can be obtained. Two intervals I_1 and I_2 have to be defined during the database setup. The first interval I_1 defines a period of time for which most of the data is being aggregated, e.g. a daily period. For the performance evaluation, a finer granularity is necessary, resulting in an interval I_2 for the performance, e.g. an hourly view. After initial assignment, the intervals cannot be changed.

All data should be put into the database in a form that allows for maximum precision. For instance, the percentage calculations should be based on the stored absolute values. The change into percentages will be done during the retrieval and output phase, e.g. by a matching SQL statement. Almost all the data examined needs a similar set of sums to be stored into different tables:

- Number of requests (without dimension).
- Size of the matching objects (dimension: byte).
- Amount of request hits as absolute value (without dimension).
- Amount of volume hits as absolute value (dimension: byte).
- Duration (dimension: milliseconds).

Although the data will not be stored in any order within the database, the output needs to match contradictory sorting criteria. For most administrators, two alternative kinds of views are believed to be of use:

1. Sorted by the number of requests, and
2. Sorted by the size of the objects transferred.

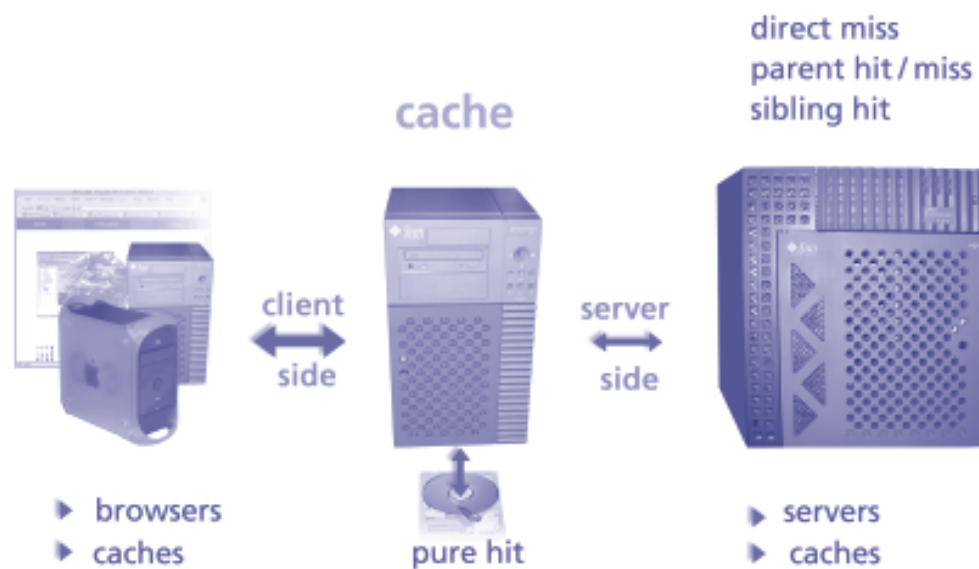
The duration mentioned above can be used to calculate the bandwidth requirement of the transmitted data. However, one has to be aware that the duration from a Squid log file is a highly inaccurate value, and thus results based on calculations

containing this value should be regarded with suspicion. The value stored into the database is usually a sum of single durations. Unless mentioned otherwise, the sum is independent of hits or misses.

2.2 Taxonomy

As shown in figure 2.1, three kinds of traffic can be observed. All queries to the cache are logged. Thus, the traffic from the *client side* matches the complete traffic as seen in the log file. The difference to the *server side* traffic is the volume saved by the cache.

Figure 2.1:
Traffic
relations



The well-known Calamaris [8] log file processor terms the *client side* traffic as "incoming". Using a different perspective, the direction of the data stream can also be viewed as flowing from the source, the web server, to the sink, the browser. Hence, from this point of view, one would call the traffic leaving the cache "outgoing". In order to reduce misunderstandings, the neutral phrase *client side* will be used in this document.

In order to avoid further misunderstandings, a few more phrases are defined here. For instance, the client querying this cache can be either a browser or another cache. Dependent caches are sometimes called *children*.

The *server side* sees only that part of the traffic which cannot be satisfied by the cache. At the server side, queries can either be sent to other caches or to the origin site. When talking about cache hierarchies, the other caches can either be on the same hierarchy level as the current cache, or they are on a higher level upward in the hierarchy. Both kind of caches are called *neighbours* (Squid calls it *neighbor*, using the US spelling) or *peers*. In order to distinguish the hierarchical level of the queried cache, caches on the same level of the hierarchy are called *siblings*. Caches on a higher level are called *parents*.

Due to the frequent changes of the names used for different hits and for the hierarchy codes, the definition of both should be kept variable, preferably as part of

a configuration file. Thus it is hoped to maintain a certain degree of upward compatibility and flexibility in order to cope with future changes. A variable definition should also simplify the parsing of foreign log file formats. Erroneous requests should be recognized by their HTTP status code, as Squid-2 no longer logs the special ERR code. The scheme of keeping the definitions variable can also be used to accommodate particular needs in the definition of "what is a hit" which may vary from administrator to administrator.

Although Squid-1 is no longer supported, it still has a considerable user base. Therefore a certain degree of downward compatibility seems desirable. However, Squid-2 will be the default.

2.3 Mandatory Sections

Of importance is all data that enables the administrator to look at a related group of caches as a whole. The report should account for the amount of traffic flowing into the group, the amount going out of the group and the amount of traffic generated by the group itself. In addition, inter cache communication may use some bandwidth, and this needs to be looked at. On the other hand, an administrator of a downstream dependent cache might be more interested in how much bandwidth or latency a cache saves him. The sections shown in this chapter constitute minimum requirements.

The server side traffic can be viewed as three different kinds of traffic:

1. traffic going *directly* to the source;
2. traffic travelling via a *parent* cache; and
3. traffic using *sibling* caches.

2.3.1 Peak values

The peak values are maintained in order to give a rough overview of the general behaviour of the cache. (Note: Peak values are not the same as the classical Calamaris-1 approach for finding peak values.)

- Distinction between TCP and UDP traffic.
- Distinction between request count, object size, and time needed.
- UDP and TCP HIT ratios.
- Hierarchy information on *direct*, *parent*, and *sibling* requests.

For reasons of backward compatibility the `ERR_*` traffic should be counted as is seen fit.

The results generated are presented as bar charts. The diagrams are intended as an executive summary. The distinction between size and requests as seen in Calamaris-2, as well as the further distinction between direct, parent and sibling traffic also seems desirable.

2.3.2 Domains

This section focuses on TCP traffic. Many cache administrators express an interest in the domains for which their cache is queried. From the 2nd-level domains the top-level domains (TLD) could possibly be generated using appropriate SQL statements. However, the almost unlimited number of 2nd-level domains forces one to store only a subset of the results. For this reason, the TLDs cannot be determined from the 2nd-level domains.

The two artificial top-level domains `<numeric>` and `<error>` were introduced. The `<numeric>` domain is used for dotted quad destinations and single big integer destinations. The latter may not be truly portable, but are frequently seen in cache log files. The `inet_aton()` of many operating systems can automagically convert single big integers into correct Internet addresses. The `<error>` is used for all unparseable domains, and for all top-level domains which were not explicitly configured. Since URLs are basically user input, they are prone to errors, and should be regarded with suspicion.

2.3.3 MIME types

This section focuses on the client side TCP traffic. Output is sorted by the rules laid out in the introduction. As with the domains, the MIME types and subtypes suffer a similar proliferation at the 2nd level. For this reason, the counted data is limited to those types and subtypes specifically mentioned in the configuration file. Any media type or subtype not listed in the configured file is counted as `<unknown>`.

2.3.4 Request method

There are different methods to access an object on a web server, as laid out in the respective standards. At the time of writing, Squid-2 knew about the following methods: `GET`, `HEAD`, `POST`, `PUT`, `PURGE`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT` and `ICP_QUERY`. The latter is used for inter cache communication using the ICP protocol. Most of the methods are non cacheable. Unknown methods are logged as `<unknown>`.

2.3.5 Overview of the requests (client side)

This section answers the question "who asks me". It will look at the complete traffic in a rough overview, sorted by the HIT or MISS status. For TCP, a further distinction into HIT, MISS and ERR is used. Please note that queries can originate from browsers as well as caches. Queries from browsers and caches are collectively referred to as querying instances.

2.3.6 Generated traffic (server side)

This section focuses on the question "whom do I ask". It looks at the TCP traffic generated by this cache. Usually, caches at the top level of a hierarchy go to the source in a direct fashion. A certain percentage of the queries can also be answered with the help of peers. Thus, a distinction between *direct*, *parent* and *sibling* traffic is necessary.

Additionally, for each peer the way an object was retrieved from the peer must be distinguished. Usually, an object will be found on a peer employing an ICP query, and transferring the object via HTTP. As new inter cache communication protocols

arise, peer objects can also be found with the help of those protocols, e.g. cache digests. Further protocols like the Cache Array Routing Protocol (CARP) are also supported. Due to the variety of inter cache communication protocols the generated traffic issue must be kept versatile in order to match future methods of inter cache communication. Also note that the traffic generated by inter cache communication is usually only recorded in the log files of the peering caches, not the log file of the retrieving cache.

Two different views are possible. One is sorted by the hierarchy code as returned by Squid. The other view additionally prints statistics for all peers queried. Essentially, the same basic table is being referred to.

2.3.7 Detailed information on querying instances (client side)

In a similar way to Calamaris-1, the traffic generated by each querying instance should be listed separately. It might be possible to combine this view with the one mentioned in section 2.3.5. It is of interest to separate the TCP and UDP traffic and to further separate the TCP traffic into HIT, MISS or OTHER. Once again, the problem of vastly growing tables is noted, but not addressed at this point.

2.4 Extensions

Whereas section 2.3 lists the mandatory minimum requirements, extensions constitute those features which are "nice to have".

2.4.1 Destination Autonomous System (AS)

When going directly to the origin site, it might be of interest to know which border gateways of the network were used, and to what extent. The destination AS number can in turn be resolved into the border gateway address with the help of routing protocols outside the scope of this project. The yield is the amount of traffic on the external links of the network.

In order to determine the AS number, the host name of the destination first must be resolved into a numerical address. When converting the host address into a class C network address, the network address can be resolved with the help of a local whois mirror and thus turned into an AS number.

2.4.2 Distribution of object size and request duration

Looking at the TCP traffic for the three classes HIT, MISS and OTHER, the object size and request duration can be summed up using fixed sized classes. The classes are calculated using the logarithm to the base of 2, although the zero is a valid input, and thus needs special treatment. Three kinds of diagrams can be generated from this statistical distribution:

1. Number of objects over a size distribution (two dimensions).
2. Number of objects over a time distribution (two dimensions).
3. Number of objects over a combined size (x) and time (y) distribution (3D).

The problems arising from inaccurate duration as seen in the log file are noted.

2.4.3 Protocols

Of interest for all TCP-based queries is the gateway protocol as seen in the URL fed to the cache. The protocol is the first part of the URL before the "://", as defined in RFC 2396 [9].

Like the tables in sections 2.2 and 2.3, it is possible for this table to grow without bounds. A possible solution might be to limit the number of protocols parsed, and record all valid names into the configuration file. All otherwise unknown protocols are counted as <unknown>. As a standard the protocols `telnet`, `ftp`, `http`, `https`, `cache_object`, `news`, `nntp`, `wais` and `gopher` should be part of distinct sums.

3 Implementation

The Seafood log file analyser is known to parse Squid-2 `access.log` files. Older versions of Squid might be supported, but were not tested. Adaptation to older log files only needs some changes to the configuration file. If you are familiar with the Calamaris [8] log file analyser, you will recognize the textual output format.

3.1 Compilation and Installation

Please refer to Deliverable 2-3 [12] for details on the compilation and installation process. Seafood is known to be compilable by the GNU c++ 2.8 series, the new g++ 2.95, or a native C++ compiler on a Unix platform which understands about `mutable`, `bool`, for-scoped loop variables, templates and essential STL constructs [10]. Egcs might do the trick, too. Seafood was successfully compiled and tested on the systems mentioned in table 3.1. With Irix, only the native compiler will work.

Table 3.1:
Systems and
compilers
known to
work

System	Compiler
Solaris 2.6, 7	SUN Workshop Compiler 5.0, g++ 2.8.1
Irix 6.2, 6.5	Irix CC 7.2
Linux 2.2 w/ glibc2	g++ 2.8.1, g++ 2.95.1

The Seafood log file processor can read compressed files. Compression is an option you can turn off during compile time. Seafood is also capable of detecting the number of CPUs currently online, **and** will use an external decompressor in a separate process, if there is more than one CPU available, and if the input is seekable. The decompressors `gunzip`, `bunzip2` and `uncompress` will be searched for using the run-time `PATH` environment variable.

The environment variables will be used from the parent process with the exception of the locale, which will be set to "C" at the start of Seafood. Using the internal decompressor is useful for single CPU machines, and for trying to decompress when reading from pipes or any other non-seekable sources.

Seafood makes use of GNU software. For instance, the `Makefile` is written for a GNU-style `make`. For the parsing of the configuration file, `flex` and `bison` are used. The standard Unix variants `lex` and `yacc` might or might not do the job; they were never tested. Past experience showed that it is just too difficult to accommodate every possible flavour of `lex`.

3.2 Configuration

If you are planning on parsing Squid-2 log files, you will not need to make many changes to the `seafood.conf` configuration file. However, please read the comments in `seafood.conf`, and configure to match your needs. There are probably a few things you would like to change, e.g. the location of your nearest whois mirror. If you are located in Europe, you might want to try the RIPE whois service [15].

Install the binary Seafood at a place of your convenience. Currently, you need to store the configuration file `seafood.conf` into the same directory as the Seafood binary. Alternatively, you can use the `-f conffile` command line option to supply a different location of the configuration file. Table 3.2 shows the currently understood configuration options.

Table 3.2:
Syntax of
all currently
known
configuration
options

Option	Arguments	Defaults
<code>no_ident</code>	<bool>	false
<code>debug_level</code>	<int> [, <int>]	0,0
<code>prefer_datarate</code>	<bool>	false
<code>peak_interval</code>	<int>	3600
<code>daily_interval</code>	<int>	86400
<code>warn_crash_interval</code>	<int>	1800
<code>show_distribution</code>	<bool>	false
<code>log_fqdn</code>	<bool>	true
<code>dns_cache_file</code>	<string>	"/var/tmp/dns"
<code>dns_positive_ttl</code>	<int>	2592000
<code>dns_negative_ttl</code>	<int>	604800
<code>dns_program</code>	<string>	"dnshelper"
<code>dns_children</code>	<int>	16
<code>dns_length_tlds</code>	<int>	0 (unlimited)

Option	Arguments	Defaults
dns_length_slds	<int>	0 (unlimited)
client_length_udp	<int>	0 (unlimited)
client_length_tcp	<int>	0 (unlimited)
irr_cache_file	<string>	"/var/tmp/irr"
irr_server	<string> none	"whois.ripe.net"
irr_positive_ttl	<int>	2592000
irr_negative_ttl	<int>	604800
irr_children	<int>	8
irr_program	<string>	"irrhelper"
irr_length_asns	<int>	0 (unlimited)
method_list	<id list>	<i>see below</i>
warn_unknown_method	<bool>	false
hierarchy_list	<id list>	<i>see below</i>
warn_unknown_hierarchy	<bool>	false
status_list	<id list>	<i>see below</i>
warn_unknown_status	<bool>	false
scheme_list	<id list>	<i>see below</i>
warn_unknown_scheme	<bool>	false
mediatype_list	<string list>	<i>see below</i>
media_subtype	<string> <string list>	<i>see below</i>
domain_list	<string list>	<i>see below</i>
warn_unknown_tld	<bool>	false

The configuration file uses a Squid-like style. The grammar is (almost) format free and case sensitive. Comments are started with a hash (#) character and extend to the end-of-line. All configuration items need to be terminated with a semicolon - this is more like Pascal than C. The <bool> type may either take the value true or false regardless of the case. The <int> type should be clear. <string> is a

double quoted string which must not extend over line boundaries. A quote character might be introduced into the string by prefixing it with a backslash. A backslash must for that reason also be escaped by a backslash.

For further details on the configuration syntax and configuration details, refer to Deliverable 2-3 [12]. Some configuration items contain list values. Lists also include an aliasing feature which allows the creation of a kind of symbolic link to a previously mentioned list item. Some lists are rather specialized:

- The method list is by default empty, but the `seafood.conf` file contains all methods known to RFC 2616 (HTTP/1.1 - [16]), Squid-2 (`NONE`, `ICPQUERY`, and `PURGE`) and those extensions used by RFC 2518 (WebDAV - [17]). You should use the methods from the supplied configuration file. There are no modifiers or aliases used.
- The hierarchy list is by default empty. The configuration file should list all possible values for the hierarchy tag in column 9 of the `access.log` file. Additionally, each item may have a modifier *direct*, *peer*, *parent*, or *none* appended to it. Some people prefer to count a `PARENT_HIT` the same as a `PEER_HIT`, because there is virtually no difference between those two. You can configure the appropriate list item to suit your needs.
- The status list concerns the status tag from column 3 of the `access.log` file. The supplied configuration file lists the status items known to Squid-2.2. Each item must have one of the modifiers *tcp*, *udp* or *none*. Additionally, each item considered a hit should have the *hit* modifier attached to it, too. Please note that not all status tags ending in hit imply **not** going to the origin server. For the very reason that your notion of a hit might be different than mine, hits are configurable.
- The scheme list is by default empty. The configuration file supplies a string list with reasonable values. You should include the *error* scheme in order to be able to distinguish between Squid generated errors logged with the scheme name `error`, and other kinds of error like unknown schemes.
- The media list of the configuration file contains the media types assigned by the IANA, and some additional media types seen in local log files. The media type is the part before the dash in the HTTP content type header. As this value is basically user input (e.g. author of web pages and server side includes), many strange things can be seen here. For the reason of media types being user input without Squid sanity checks, a warning option was not deemed feasible (yet).

For each media type, you can define the sub types you are interested in. Each media type may have only one sub type list attached to it. If you use more than one, currently the new list will overwrite the old list. All sub types not mentioned will be counted as sub type `<unknown>`.

- The domain list contains all top-level domains currently known. Please check that none are missed. The virtual top-level domain `<numeric>` counts all those URL hosts entered as dotted quad or as big integer numbers. The virtual top-level domain `<empty>` represents malformed URLs. The `<unknown>` value will be used for anything not in the list.

The configuration options `log_fqdn` and `irr_server` control the use of external look up services. If you set the Seafood option `log_fqdn` to true, the clients in the client request tables will be printed symbolically. While most clients are usually local, this will take a small amount of time.

The option `irr_server` has got a high time consumption. If you are using AS-based look ups, you should have applied the `log_ip_on_direct` patch [19] by Henrik Nordström [20]. Otherwise, the destination address must be looked up before the AS number can be determined. Also, the whois server might be quite slow. Setting the `irr_server` to the value `none` disables the AS look up feature.

Both, DNS look ups and AS look ups will be cached for later use. The cached information resides in main memory during the runs, and will be stored in separate GDBM files between runs. Please note that the GDBM caches can grow quite large.

The interval any DNS and AS look up will be cached can be determined separately for DNS and AS as well as for successful and failed look ups. Due to the locking nature of the look up calls, external helper programs `dnshelper` and `irrhelper` are started. The number of these helpers can be configured separately for DNS and AS look ups. The number of helper processes is limited between 1 (minimum) and 255 (maximum), with some further limitations imposed by the operating system.

There exist five kinds of results that have an abundant or nearly unlimited amount of data: the top-level domains, the 2nd-level domains, the UDP clients, the TCP clients and the AS results. Those results can be limited in their number of entries, using a configuration option. The default is not to limit any result.

The-top-level domain results are further limited to the number of explicitly configured domains. The number of 2nd-level domains is nearly boundless, and for a typical week day on our cache servers, the number of 2nd-level domains found can exceed 30,000. In a multi-level caching hierarchy, the top-level caches may only see a small number of sibling caches, and thus produce small client results. But local university caches usually see all browsers from almost all computers in the network. Therefore, a limitation of the client tables is necessary. The AS limitations are only taken into account, if an AS result will actually be produced.

3.3 Running Seafood

Seafood can be run on one or many of your log files. Each run will summarize all log files poured into it. Each log file may be plain, gzip compressed or bzip2 compressed. Before running Seafood, check your configuration file, and make really sure that you have configured all the options you want.

A typical run, assuming a Bourne or compatible shell, would look like this:

```
$ ./seafood /some/where/access.log result 2 errors &
$ tail -f errors
```

On `stderr` warning messages, informational messages and other material will show up. The informational messages start out with a hash # character. All actual warnings start out with the line number of the offending log file line first.

The messages start out with the run-time endianness detection for the netmask code. Only big- and little endian architectures are recognized. Next the operating system imposed limits on file descriptors and number of processes are checked, in order to determine the number of external helper programs. In the third step, DNS and AS caches are connected to their respective GDBM file, and read into the main memory. If using compressed input on multi-CPU machines, an external uncompressor will be started. Single CPU machines will use an internal decompression function. Please note that the uncompress algorithm is not available as an internal decompressor. The message above is from a two CPU machine. Further informational messages are a kind of progress indicator. They may be interrupted with actual warnings.

```
45643: neither TCP nor UDP, counting as TCP
+ timestamp=933546744, duration=3,
client="xxx.xxx.xxx.xxx"
+ status="NONE/400", size=2406, method="GET"
+ url="http://2.htm"
+ ident="-", hier="NONE/-", mime="-"
```

The above message is a warning about line 45643. The log line constitutes neither TCP nor UDP, but will be counted as TCP. It is just the informational warning that your results may be off, though in this case, counting as TCP is correct.

After the log file has been processed, the output will be generated. Each section of the output has a log line associated with it. Since there can be textual tabular output as well as a database intermediate file, you might see the output progress indicators twice, once for each output format. If the DNS and IRR helpers were used, the internal cache contents are dumped into their respective textual database files. DONE signals the finish of the output.

Do not be disturbed if Seafood seems to sit waiting on one of the positions mentioned in table 3.3.

Table 3.3:
Possible
reasons for
slow output

Position	Reason
2lds...	Seafood is sorting several tens of thousands of domains. On slower machines this may take a while.
asn...	If you configured to use IRR, Seafood is querying the IRR. Since the IRR server may be slow in answering, up to 30 s per request, this may eat a considerable amount of time. Also, if your destination is not logged as dotted quad, the destination must be looked up before its AS number can be determined.
clients...	Seafood usually reverse resolves the client IP address into something symbolic.

3.4 The Textual Results

There are a number of textual tables written to `stdout` after a successful parsing. Due to possible look ups, the generation of the sorted tables may take some time, too. If you are familiar with Calamaris [8], you will recognize the output format as being similar:

- The overview section contains a simple table just displaying the sums of all TCP, all UDP and SUM traffic, including hit rates.
- The status section contains a UDP table, which is sorted by HIT and MISS, and a TCP table, sorted by *hit*, *miss* and *none*.
- The hierarchy section displays an overview of the server side connections with the different hierarchy tags sorted by *direct*, *parent* and *peer*. Please note that even though *peer* includes mostly sibling, for some hierarchy codes it also includes parents. The tag lines contain a hit rate, which is nonsense, and serves as a validator for your configuration file. The second hierarchy is sorted by the peer or parent contacted, and contains the hierarchy tags used with that particular peer.
- The hierarchy information is followed by the request method table. RFC 2518 (WebDAV - [17]) is part of the configuration file, and will thus be counted. Unknown methods are counted as `<unknown>`.
- The URL scheme is part of another table, sorted by requests. Unknown schemes are counted as `<unknown>`.
- The top-level domains as configured are displayed in the following section. The first table is sorted by requests, the second one by volume. The number of domains shown can be limited to the top-N.
- The 2nd-level domains are displayed in two tables. Again, the first table is sorted by requests, the second one by volume. Both tables can be limited in their extent to a top-N chart.
- If you configured an IRR server, your *direct* hierarchy destination hosts will be grouped by the destination Autonomous System Number (ASN) the server resides in. Two tables are generated, sorted by requests and sorted by volume. The virtual entry `<NOIRR>` implies that for the given host, an origin AS could not be determined. Both tables can be limited to the top-N.

The AS information will be obtained by querying an external helper process called `irrhelper`. For each request, the helper process opens a connection to the configured whois server, sends the query, parses the answers and closes its connection. Any complete whois accessible mirror of an IRR can be queried.

Waiting for any non-local whois server is really slow, up to 30 seconds per request. For instance, processing a log file took 12 seconds, and waiting for the whois server to answer 705 seconds. Using your own mirror is recommended, instructions and software can be found at the RADB site [21]. Such a whois mirror server can also be used for Squid: see `as_whois_server` in your `squid.conf` file. Refer to the ACL schemes `dst_as` and `src_as` respectively.

- The media types and sub types are a run-time configurable option. Again, two tables, sorted by requests and sorted by volume, are the output. The tables are sorted primarily by the media type sums, and secondarily by the sub type accumulation.
- The textual output contains a summary of ports that were used. In order to reduce the output volume, the ports are grouped by 1024 ports. Port 80 is mentioned separately.
- The UDP client side table is just sorted by requests. The table displays the UDP traffic and the part of the *hit* traffic.
- This is a little awkward table. The contents are sorted by requests of the client, and for each client, the *hit*, *miss* and *none* amount is shown.
- The performance data are associated with the term *peaks* for historical reasons. For each `peak_interval` from the configuration file, all requests during that interval are grouped and displayed consecutively. The first table deals with requests, the second table with volume. The third and final table deals with the request duration, even if you configured to use bandwidth for the other tables.

The first three columns display the UDP sum, the UDP hit amount and the relative UDP hit : UDP amount. The next three columns do the same for TCP. The final six columns deal with *direct*, *parent* and *peer* traffic, and the relative number based on the TCP amount. Please note that the sum of the last four percentage columns in a line do not yield 100. The missing part is the *none* traffic, which is not logged (yet).

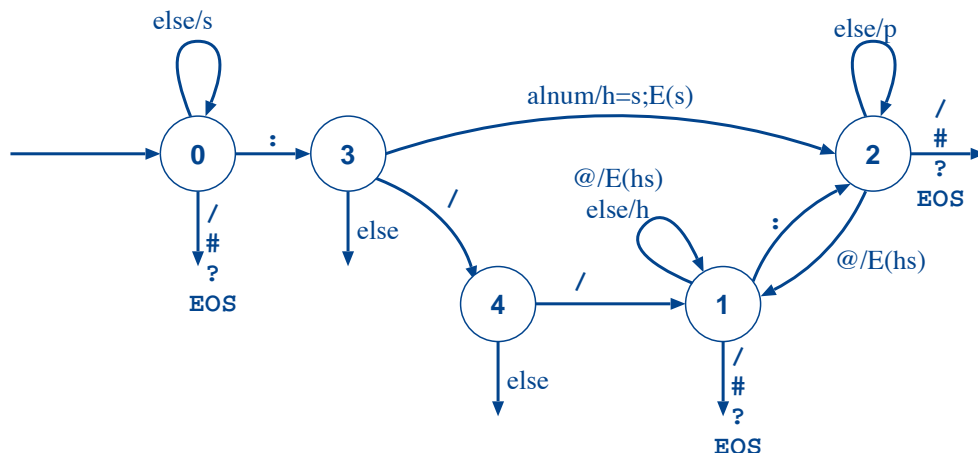
The duration table omits the UDP columns, as ICP requests are usually logged to take no time - even if that is not true, and even if other log software like Calamaris assumes it to take a microsecond each.

- If you configured to show the distribution tables, three further tables will be part of the result. The first table shows the distribution in time and size of TCP *hit*, the second of TCP *miss* and the third of TCP *none* requests that include some errors. The interior of the tables is the distribution in time and size. The final sum row of each table is the accumulated distribution in size, and the final sum column of each table the accumulated distribution in time. A special program can be used to generated 3D diagrams and VRMLs from the data.
- The final table displays some statistics about the analyser, e.g., how much time it spent in the parser loop and how much time it needed overall. Of interest may be the lps value in parenthesis, which is the lines-per-second amount, parsed.

3.5 Parsing a URL

The URL is a user entered input, and may thus contain grossly malformed input. Figure 3.1 shows the finite state automaton which extracts the method, host name and port from a URL. Start state is state 0. The final state 5 is not explicitly shown.

Figure 3.1:
Finite state
automaton
for parsing
URLs



The connotation at the arcs may look a little weird. My apologies for not using standard notation. The Mealy automaton performs its actions during the transition to a new state. If the arc is only labelled with a single character, this character will be eaten without doing anything with it. Otherwise the action is associated with the mentioned character or character class, and separated by a slash.

The character class `alnum` contains alphanumeric characters. The virtual character class `else` contains all characters which are not any other arc leaving the node. The action is abbreviated, too. The action `s` means to add the character to the scheme, `h` to add to the host and `p` to add to the port. The action `E()` empties the arguments.

The numbering of the state nodes is arbitrary and was chosen in order to simplify the implementation of the automaton. The weird arc from state 3 to state 2 was added in order to be able to parse the host name and port of tunnelled connections, e.g. to be able to parse the following log line:

```
[...] CONNECT some.host.domain:443 - DIRECT/some.host.domain -
```

3.6 Selected Internals

This section will deal with a few choice internals. For more implementation details, please refer to appendix C.

Seafood is capable of reading plain textual input and compressed input in a variety of formats. If more than one CPU is available, decompression takes place using an external decompressor that is run as an input filter. If there is only one CPU available, decompression is achieved internally using appropriate library calls. Compression formats understood are `gzip`, `bzip2` and `compress`. The latter is only available on multi-CPU platforms. Due to its design, Seafood can be extended to use decryption.

The parsing API of the Seafood software provides high-level functions to return different integer flavours, floats and words from the input stream. In order to achieve maximum speed, care was taken to touch each character only once during parsing.

The symbol tables which contain status tags like `TCP_MEM_HIT` or hierarchy tags like `CD_PARENT_HIT` are dynamically generated during run-time from the information supplied in the configuration file `seafood.conf`. For efficiency reasons, the symbol tables used for matching are implemented using trie data structures [28].

When comparing the match function of a trie with a generic hash table, both need a time proportional to the length of the word looked at. But as a trie by then has arrived at the stored (token-) value, a generic hash needs to check its collision avoidance algorithms first.

When looking at the matching speed of words not in either container, the trie excels even further. As soon as the first letter not in the trie is encountered, the trie will 'know' a mismatch. A generic hash table on the other hand will need to touch all letters of a mismatch and possibly do some collision avoidance in order to conclude a mismatch.

The drawback of a trie is that the insertion of new elements takes considerably more time than insertion into a generic hash. Therefore, tries are used for static information which is read in once from the configuration file during start-up, and which is just matched against, while hash maps are used to store dynamic content found during the parsing of the log file.

When porting the first Perl prototype to C++, as high level a construct was needed as Perl supplies to its users. The necessary constructs included associative arrays which could be symbolically indexed, excessive use of regular expressions called and the basic data types of a `Strings`. Still `char*` pointers and character vectors are kept for performance reasons. The string classes are primarily used for indexing associative arrays and returning symbolic information from functions. All maps are indexed by a `String` and may use an arbitrary value class.

4 Database Design

4.1 Database Implementation

All table names start with the `sf_*` prefix in order to avoid cluttering with other tables already defined in the current namespace. Figure 4.1 shows the physical model of the set of tables which depend on the daily interval I_1 . Due to the less abstract notation, Postgres data types were used instead of SQL types. As mentioned during the database design phase, the volume results should be stored using at least 64-bit integers. The design was optimized in favour of less tables, as it is easier to select and group results from a single table than to aggregate from different tables. A single character was chosen as Boolean type for predicates. Some tables contain redundant data in order to simplify post processing of the data.

During development Oracle 7.3, Oracle 8i and PostgreSQL 6.3.* were used. Other database products were also considered during the design of the tables and relationships. The schema can easily be converted to different databases.

Figure 4.1:
Physical
model of
tables depend-
ing on the
daily
interval

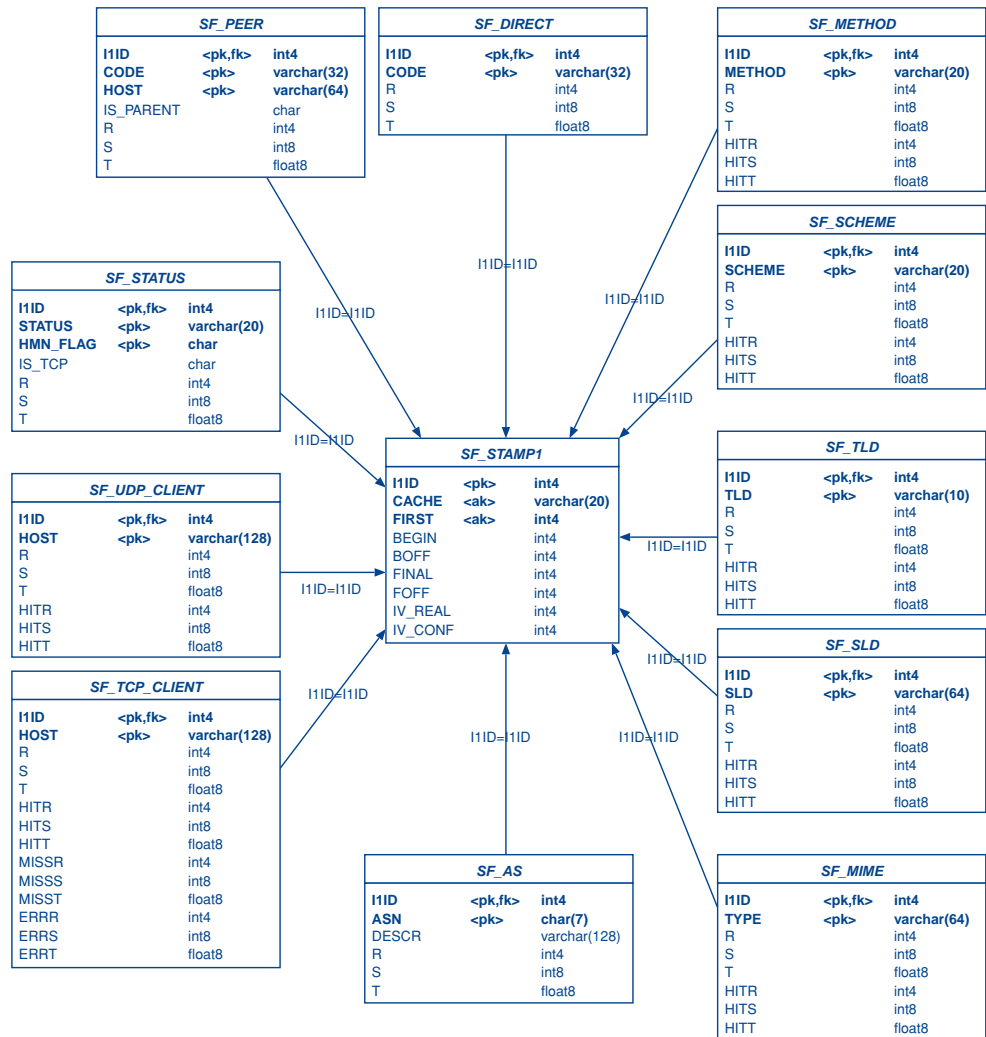


Figure 4.1 shows a circular arrangement of tables around the central table `sf_stamp1`. The primary key fragment `I1ID` is an abstract number. Table `sf_stamp1` uses an alternate key consisting of the combination of the cache host name and the *beautified* start time stamp `FIRST` of interval I_1 , the real key for the central table `sf_stamp1`. The real start time stamp is contained in the `BEGIN` column. The `I1ID` provides an *efficient* glue between the tables shown in figure 4.1.

Due to the fact that there does not exist *generic* support for sequences, row identifiers, or triggers, the interval id generation is provided by the insertion script. If the database product allows, the dependency of the outer tables' interval id on the central id should be modelled using a foreign key or a `references` constraint on the outer tables' id column. A redesign should focus on a powerful generic abstract database and leave these details to be programmed, if necessary, in the database actually used. In this regard, the database design is neither as optimal as it could be nor as abstract as it should be.

The central table `sf_stamp1` contains the timestamp information about the daily interval for which Seafood was run. It also uses the name of the cache it was run for, or the logical name of the cache group. Part of the table is also the offset to UTC in order to provide daylight savings transition information for output processes. Finally, the interval for which the sum was run, and the configured interval are part of the central table.

All tables on the ring contain the (R,S,T) triple with the following meaning:

- The R part is the request count.
- The S part is the volume sum in bytes.
- The T part is the time spent in seconds, as reported in the log file.

Please note that the UDP related data usually has no noticeable transfer time, at least from the view of the log file.

If one is using a different database product, e.g. one which supports native arithmetic types, R and S should be assigned eight byte integers, and T an eight byte float. Even though Seafood only uses a four byte integer for its internal request counters, the database may focus on a much larger interval. For this reason, the sum of a year's worth of requests might overflow some internal database registers, and some database products report the wrong results instead of an error.

Some tables contain the related (HITR,HITS, HITT) triples. These triples refer to the number of hits as perceived by Seafood. Please note that a hit triple always contains values less or equal to the plain (R,S,T) triple in the same table and row. Furthermore, the `sf_tcp_client` table contains a (MISSR,MISST,MISST) triple for misses and an (ERRR,ERRS,ERRT) triple for errors and those non-hits with a hierarchy code of NONE.

Starting on the left side of the ring, generic TCP and UDP counts are combined into one table `sf_status`. The glue and the status code themselves are not sufficient keys when trying to distinguish misses from errors, e.g. a TCP_MISS status might be the answer to a miss as well as an error. Therefore, the `HMN_FLAG` needs to be part of the key. The `IS_TCP` predicate on the other hand is a piece of redundant data, as the status code sufficiently distinguishes between the transport protocols. A future design might keep the status code in a separate table, and assign the predicate a stored procedure.

The top of figure 4.1 shows the tables dealing with the server side traffic of the Squid cache. The hierarchy code is part of the primary key. For a sibling and a parent, the host contacted is also part of the key. Siblings and parents are collapsed into one `sf_peer` table. The `IS_PARENT` predicate is redundant again, because the hierarchy code sufficiently distinguishes between parent and sibling. When going directly to the origin site, the destination host is neglected in the database, even though Seafood knows about them internally.

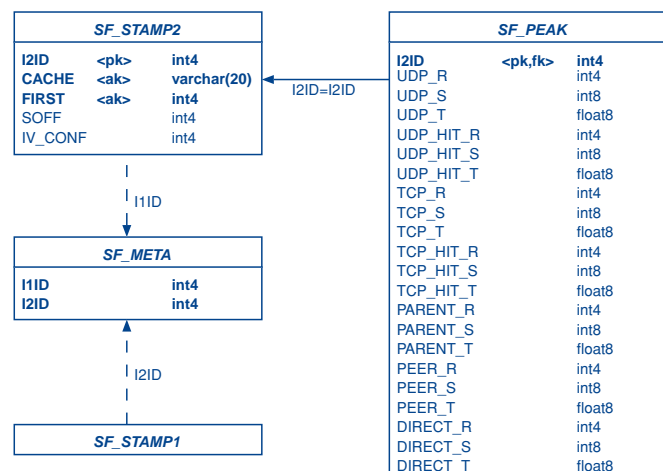
On the right side of the ring, the request method, the scheme, the top-level domain, the 2nd-level domain, and the MIME types are counted. All five tables contain the hit count besides the plain count. Part of the primary key of the method table, the scheme table, the top-level domains and the MIME types are only those items which were configured in `seafood.conf` by the time Seafood was run. Furthermore, the number of entries in the domain tables for a particular cache host and timestamp are limited to the top-N domains as configured with Seafood. The top-level and 2nd-level domain tables could theoretically be combined. But as the administrator is expected *only* to store the top-N 2nd-level domains, the top-level domain information is not completely contained in the 2nd-level domain table.

The direct hosts can be combined into networks and autonomous systems, if a whois service is available, and Seafood was thus configured. The `sf_as` table contains the destination AS number of origin servers which were visited directly. Additionally, the description for the AS as seen during the Seafood run is part of the database. There is no hit information to the `sf_as` table, because the destination AS information is obtained from the direct fetches.

Finally, the client tables `sf_udp_client` and `sf_tcp_client` deal with the client side of the cache. Part of the primary key is the client host address. The format of the client host, symbolic or numeric, is taken as-is from the Seafood provided output. Note that the number of clients is limited to the top-N clients as configured in Seafood. Especially for the TCP clients, it seems feasible to provide a cut-off threshold of some percentage of the requests in order to weed out clients whose access is denied. Please note that such an alternative is not part of the current project.

Figure 4.2 shows the `sf_stamp2` table, which deals with the hourly interval I_2 . Again the start timestamp and the cache hostname are alternate keys for the interval id. Please note that this time interval id `I2ID` is used.

Figure 4.2:
Remaining
tables



Only the performance statistics table `sf_peak` depends on the second stamp table. For each hourly time stamp, it collects the UDP and TCP data in hits and misses, as well as the hierarchy data. Not part of the table is the count for objects with the hierarchy code NONE.

Finally, the `sf_meta` table records the highest number the interval IDs currently have. The insertion script makes use of these numbers, increases them, and generates appropriate SQL statements using the new interval IDs. A redesign should assume the availability of sequences, and leave this detail to the concrete implementation for a database.

4.2 Filling the Database

The examples shown in this section are based on Oracle 7.3. However, due to the fact that a scripting language like Perl is used, you should be able to easily modify it to suit your needs. The database scripts use Perl's generic functions of the abstract DBI interface. The interface itself, using a connect string, maps the calls to the underlying concrete database driver (DBD) which communicates with the database. Java and ODBC allow for similar abstract database programming, but were not considered in this project.

In order to use Seafood, you will need to create the tables in your database, once only. Also, the meta information table `sf_meta` needs to be filled with the default pair (0,0). For Oracle, a Perl script is part of the Seafood software suite. You must modify the `create-ora.sql` script before starting it to suit the concrete database product you are using. For instance, PostgreSQL does not know about foreign keys, but is able to use a `references` column constraint during table construction.

After creating the necessary tables, data can be filled in. Seafood can create a so-called database intermediate file, which is an unsorted textual file meant to be read by an external Perl script which feeds the sums into the appropriate tables within your database. The example scripts are based on Oracle 7.3 and PostgreSQL. The script needs some environment variables and script variables to be set in order to work correctly. Refer to Deliverable 3 [13] for details.

The insertion script starts by loading the concrete database driver and connecting to the database. In the next step, it tries to verify that all necessary tables are to be found in the data dictionary of your system. Finally, it reads the database intermediate file and stores the data into the database. If an error is detected, the insertion will be rolled back and the script aborted. Otherwise, success will be reported and the changes committed.

At the moment, you are unable to insert the same file twice. A more accurate insertion should be possible, e.g. checking the first stamp and cache host from the time stamp tables. If detected, updates should be used, either by adding to or overwriting existing data. But at the moment, only insertion of new rows can be used.

In case you want to remove all tables from your database, you just need to drop the appropriate tables. Some products like PostgreSQL are not capable of cascading dependent drops, so you might need to expand the uninstall SQL script in order to drop automatically constructed indices.

Attention! Running the uninstall script will remove all Seafood related data from your database! You could lose years worth of data.

Table 4.1 shows the programs and scripts described so far. Please note that the Oracle dialect is used in all the examples. Scripts for other database dialects were also supplied, but not tested.

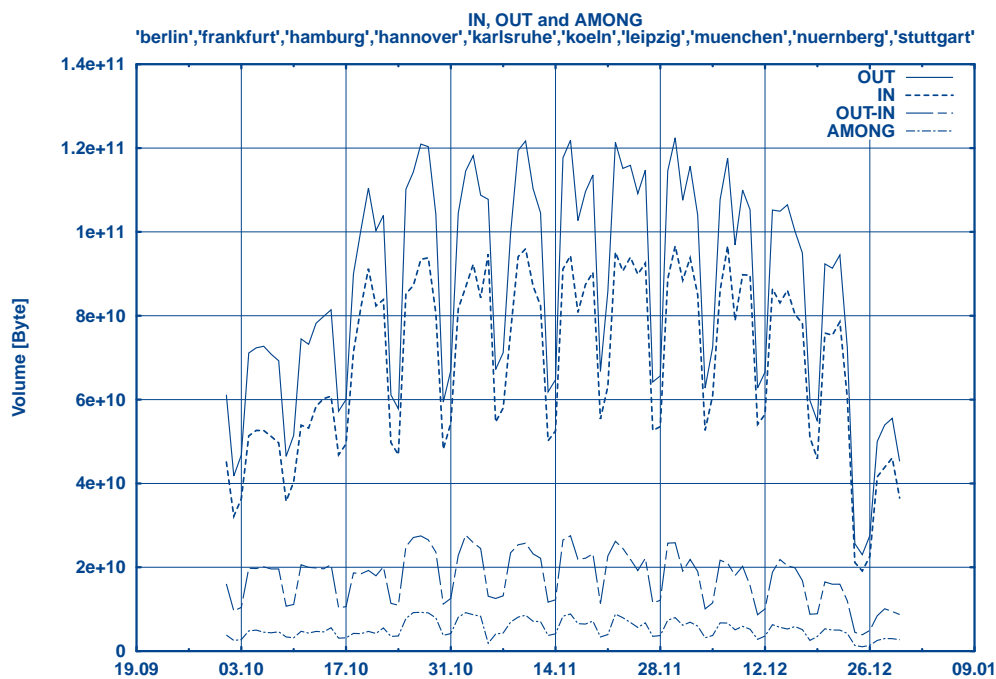
Table 4.1: Scripts and programs supplied

Program	Meaning	Language
seafood	log file analyzer.	C++,binary
create-*.sql	Database table creation script.	SQL
destroy-*.sql	Database destruction script.	SQL
insert-*.pl	Database value insertion script, which takes the results from Seafood and puts them into the database.	Perl
Tables.pm	Helper module for the insertion script.	Perl

5 Web Interface

Besides querying for the plain results of a single table stored within the database, the data of greater interest can be found by combining and grouping more than one table. Figure 5.1 is an example of a powerful combination.

Figure 5.1: Example for a combined extract from 10 cache hosts



5.1 The Selection Menu

The web-based interface is divided into six separate areas. The first two areas deal with the selection of the interval of interest. When a user requests the CGI page from a web server, the CGI script queries the database for the extent of data stored, and thus creates a default of the smallest and largest time stamp available.

Figure 5.2:
The
web-based
interface

Start Time:	00	hours,	30	.	Sep	.	1999
Final Time:	00	hours,	01	.	Jan	.	2000
Hosts:	<ul style="list-style-type: none"> berlin frankfurt hamburg hannover karlsruhe koeln leipzig muenchen nuernberg stuttgart 						
Diagram:	<input type="radio"/> small <input checked="" type="radio"/> medium <input type="radio"/> large		<input checked="" type="radio"/> GIF <input type="radio"/> EPS <input type="radio"/> PNG <input type="radio"/> AI				
Query:	Peak TCP and TCP-HIT					<input checked="" type="radio"/> by req. <input type="radio"/> by vol.	
Action:	submit		reset				

The third part lets a user select a cache host. The cache host list is also generated by a query to the database. Please note that at least one cache host must be selected. The fourth section deals with the canvas size and the media type of the output. The media types can easily be extended to any gnuplot supported graphics format.

The fifth section determines the kind of query. The prototypical implementation contains nine different options, which will be covered in the following sections. The kinds of query can easily be extended to options of other interests by extending the Perl scripts. Also, a selection by requests and by volume is part of the selection. The final column either submits the choices to the next stage, or resets the form to its default.

The following sections describe the different kinds of query which were selectable in area five.

5.1.1 Peak TCP and TCP HIT

The first kind of query in the selection menu is a simple selection from the peaks table. The peak table is kept at the finer granularity I_2 , which allows a good overview of daily or weekly behaviour comparisons of up to two caches. The selection is straight-forward: for the specified interval, a graph with the requests and hits is drawn for each selected host.

5.1.2 Non-GET method sums

The non-GET method option allows for a quick sum of all methods which were not GET methods. Please note that a more flexible interface would allow the user to specify the methods to exclude. Also, note that ICP_QUERY is excluded, as it constitutes about 75% of the queries we see, and can be determined from UDP traffic (not in the web GUI).

Within the selected interval, for each selected cache host, the sum of its non-GET queries and respective hits are graphed. The logarithmic scale was chosen to simplify the two regions of interest. The values in logarithmic scale are off by 1 in order to overcome the $\log(0)$ problem.

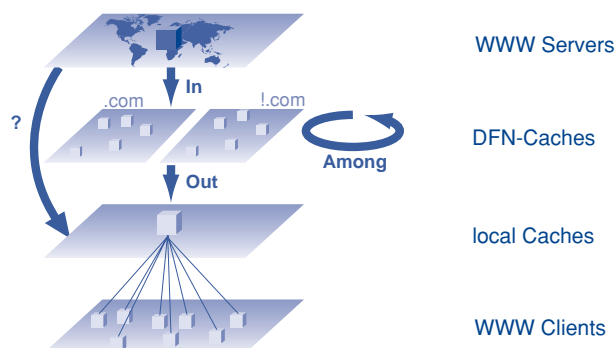
5.1.3 Special TCP clients

In order to determine the amount of traffic generated by a cache mesh as the client to itself, like fetching hits from neighbours, or obtaining objects from parents, the special TCP client option allows to select a subset from the TCP client table. A more powerful interface would allow the user to specify the set of hosts to select. For the sake of a prototype, the search set is hard coded into the query, but can easily be changed. Please note that when porting the queries to suit your needs, you should always match both the symbolic and the numeric host names.

5.1.4 IN, OUT and AMONG

The IN, OUT and AMONG options allow for a more powerful combination than the ones previously shown. It allows the evaluation of the efficiency of a cash mesh, the savings as OUT-IN, and the overhead - the traffic AMONG the caches. Refer to figure 5.1 as an example. Unfortunately, the traffic bypassing the cache mesh, labelled with a question mark in figure 5.3, cannot be computed, as it does not appear in the cache log files.

Figure 5.3:
Viewing
a cache
mesh as
a source



Please note that the AMONG traffic is - in our case - completely limited to the backbone. The information for OUT is determined from the TCP clients table with the exception of the caches participating in the mesh themselves. The information used for IN is determined from the *direct* hierarchy table. AMONG is selected from the *peer* hierarchy table, which contains both parent and sibling calls.

The savings OUT-IN are graphed separately in order to make the savings comparable to the overhead. Figure 5.1 shows the savings and overhead for the DFN caches from October to December 1999.

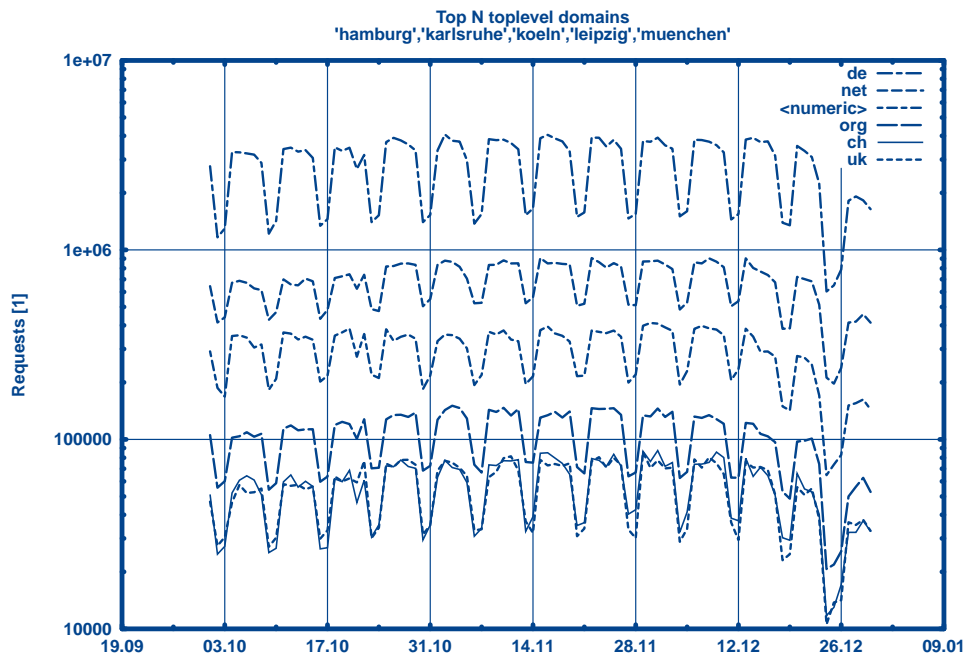
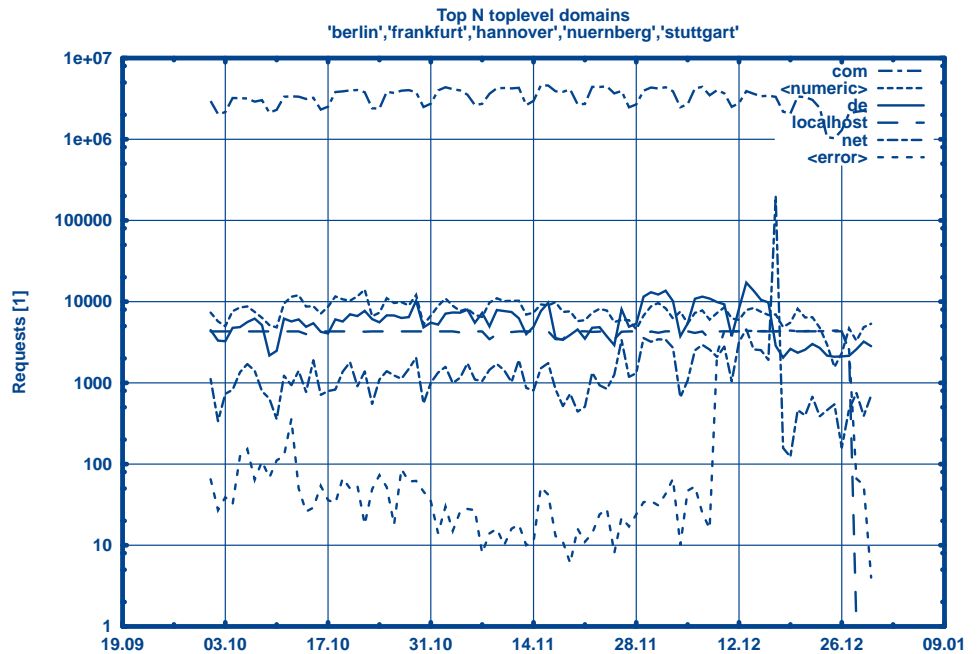
5.1.5 Frequent vs. top-N top-level domains

There are two options dealing with the top-level domain names. The *frequent* option uses a fixed set of frequently encountered top-level domain names. The *top-N* option on the other hand dynamically determines the six most common top-level domains

for the specified interval from the top-level domain table. Please note that viewing by requests and viewing by volume might yield a different set of top-N domains for an identical interval.

When viewing the examples, please note that our cache mesh is partitioned into hosts serving *.com requests and hosts serving the logical !.com domain.

Figure 5.4:
Example
for top-N
*.com and
!.com
domains
by request



5.1.6 Frequent MIME types

The frequent MIME type option selects a fixed set of often encountered MIME types from the MIME table. A more adaptable GUI would allow the user to select the types.

5.1.7 Top-N autonomous system (AS) destinations

The AS option selects the top six most frequently used destinations which were visited by the selected cache hosts. Note that the AS table is created from direct hierarchy destinations, with the destination host condensed into the AS number. Therefore the AS diagram will give some insight into the extent to which exterior links are used.

5.1.8 Top-N clients

A set of different client options allows for separate diagrams for TCP and UDP client traffic, further separated into hit and summary traffic. For the specified interval and cache host set, the top six clients are determined and plotted.

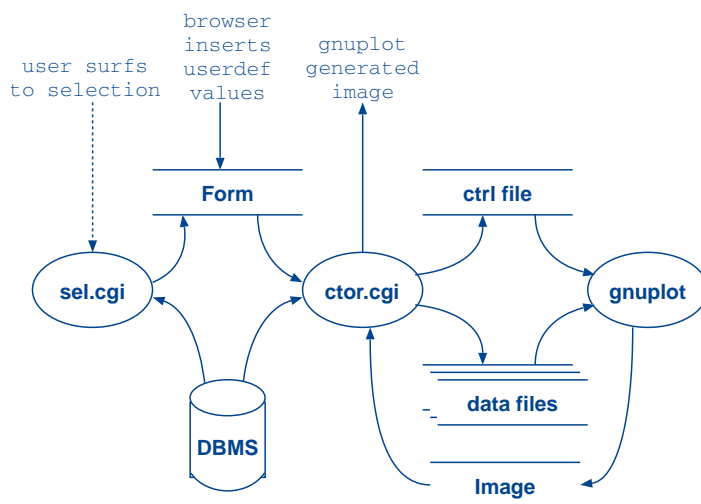
5.1.9 Hierarchy performance

The hierarchy performance, separated into parent, sibling and direct traffic, can be plotted using the last option. For the selected interval and cache host set, the traffic is plotted into three graphs using a logarithmic scale. The values could have been derived from the peaks table for the finer interval I_2 granularity, but in order to stay comparable with other interval I_1 based output, the graphs are selected from the hierarchy tables.

5.2 Implementation of the Prototypical Interface

A web-based GUI was chosen due to its implementation independent access to the database underneath. Figure 5.5 shows the data flow diagram.

Figure 5.5:
CGI data flow
of web GUI



Whenever a user requests the selector's URL, a CGI program `sel.cgi` will read the smallest and largest timestamp in the database, and create a set of possible cache hosts from the entries in the database. The HTML formatted form is sent to the user's browser, which will in turn enter the user's selections.

By pressing the submit button, the user's choice of data is sent to the `ctor.cgi` script. If the submitted data is consistent, the database will be queried, usually several times per option. The query results are constructed into intermediate data files to be fed into a modified version of `gnuplot`. While generating the data files, a control file for `gnuplot` is also created. `Gnuplot` in turn will create the image file. All files reside in a temporary directory.

Finally, control is returned to `ctor.cgi`, which then reads the temporary image file and copies it to its connection to the browser. Some cache control headers are added in the reply. All temporary files are unlinked. Care was taken to unlink these files even in the case of errors, though abnormal termination on a signal might leave some clutter.

A modified version of `gnuplot` is used which is capable of parsing UTC time stamps, and which can convert the time stamps into a human readable format. A patch is included in the source distribution, and was submitted to the maintainer.

The selection script is the front-end to the database interface. Using two queries after start-up, the script tries to determine the minimum and maximum time stamp available, and the set of cache hosts. The form selectors for the time stamps are created and defaulted to the time stamps found in the database. The rest of the script just prints the form and explanatory messages.

The constructor script is a rather monolithic creator of images. The script starts by sanitizing the user submitted values. If the values are OK, more data structures are set up for later perusal. In the next step, the script connects to the database, creates the `gnuplot` control file, and calls a work function to obtain the results of a query. After the function finishes, the database connection can be closed, and `gnuplot` will be called. If `gnuplot` returns successfully, the image will be copied onto `stdout`. Please note that the database calls are bracketed within an `eval` block in order to catch errors.

The work functions do not take any arguments. Each function is expected to return one or more `gnuplot` control statements to be written into the control file on return from the function. Each function will do one or more selections from the database, and arrange the answers in data files suitable for parsing by `gnuplot`. The data files are created in a temporary folder. The name of the data files are part of the `gnuplot` control return value from the work function.

5.3 Example for an Implementation of the SQL queries

The previous sections showed the abstract selection from tables, and the updated database design. This section will deal with the concrete implementation of the SQL queries used to obtain values from the database. All examples are shown for volume selection. The selection by request is part of the source.

Some Perl variables are visible in the queries. A better implementation would be using database stored procedures, and calls to them with variable arguments. Still, the current implementation is also flexible enough for a prototype. Refer to table 5.1 for commonly used variables. Further variables are described in the sections.

Table 5.1:
Common Perl
variables
in select calls

Variable	Content
\$start	Start time stamp of GUI selected interval.
\$final	Final time stamp of GUI selected interval.
\$cache[\$i]	One of the GUI selected cache hosts within a loop.
\$cacheset	All GUI selected cache hosts.

The example is shown by describing the action taken for the "IN, OUT and AMONG" query. It works on a set of selected cache hosts simultaneously. The output are just four graphs, showing:

1. the amount of data sent from the caches (OUT);
2. the amount of data received by the caches from external sites (IN);
3. the amount of data generated by the caches acting as a source (OUT-IN); and
4. the overhead due to inter cache communication (AMONG).

The selection of the OUT values is similar to the previous section. For all clients, with the exception of the DFN caches, the sum of all client traffic for each time stamp is calculated. Even though only some top-N client caches are put by name in the database, the "more[.]" pseudo cache contains the sum of caches not mentioned explicitly. Therefore, the OUT value is complete.

```
select st.first, sum(tc.s)
from sf_stamp1 st, sf_tcp_client tc
where st.ilid=tc.ilid and st.cache in ( $cacheset )
and st.first between $start and $final
and not ( tc.host like '%.win-ip.dfn.de\' or
          tc.host like \'193.174.75.%\')
group by st.first
```

The IN values are selected from external traffic going directly to the source. Due to the DFN caches being top-level caches of a hierarchy - they have no parents outside themselves - the IN value is complete. Users at a different level of a cache hierarchy might want to add their parent traffic. On the other hand, in many cases the direct traffic is the most expensive.

```
select st.first, sum(hd.s)
from sf_stamp1 st, sf_hier_direct hd
where st.ilid=hd.ilid and st.cache in ( $cacheset )
and st.first between $start and $final
group by st.first
```

The savings are calculated for each time stamp as OUT minus IN for the current stamp. The overhead AMONG is obtained from the sibling and parent traffic in the `sf_peer` table.

```
select st.first, sum(hp.s)
from sf_stamp1 st, sf_hier_peer hp
where st.ilid=hp.ilid
and st.cache in ( $cacheset )
and st.first between $start and $final
group by st.first
```

6 Conclusions

The project proposal refers to correlating data, trends not immediately obvious from the basic data. The goals of "data mining" are to detect, interpret and predict qualitative and quantitative patterns in the data [27]. The proposed "Extended Cache Statistics" will certainly not do all this, but it is believed that storing well-selected basic data is the first step in the right direction, and it is hoped that Seafood is able to provide at least some glimpses of what is possible with the data.

7 References

- [1] RVS WWW Server
<http://www.rvs.uni-hannover.de/>
- [2] Einstieg BMBF-Server
<http://www.bmbf.de/>
- [3] German Research Network
<http://www.dfn.de/welcome/>
- [4] Trans-European Research and Education Networking Association
<http://www.terena.nl/>
- [5] TF-CACHE - WWW Cache Coordination for Europe
<http://www.terena.nl/task-forces/tf-cache/>
- [6] Squid Web Proxy Cache
<http://www.squid-cache.org/>
- [7] Network Appliance - Network Storage and Caching Solutions
<http://www.netapp.com/>
- [8] Calamaris Home Page
<http://calamaris.cord.de/>
- [9] Uniform Resource Identifiers (URI): Generic Syntax
RFC 2396
- [10] Standard Template Library Programmer's Guide
<http://www.sgi.com/Technology/STL/>

- [11] Deliverable 1 - Request for Peer Comments
<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable1.pdf>
- [12] Deliverable 2.3 - Seafood - a Log File Analyser
<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable2-3.pdf>
- [13] Deliverable 3 - Database Design
<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable3.pdf>
- [14] Deliverable 4 - Web Interface
<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable4.pdf>
- [15] RIPE Network Coordination Centre
<http://www.ripe.net/>
- [16] Hypertext Transfer Protocol -- HTTP/1.1
RFC 2616
- [17] HTTP Extensions for Distributed Authoring -- WEBDAV
RFC 2518
- [18] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.
Design Patterns - Elements of Reusable Object-Oriented Software
Addison-Wesley, 1995.
- [19] Squid-2.2.STABLE5: Log destination IP on DIRECT patch
http://squid.sourceforge.net/hno/patches/squid-2.2.STABLE5.log_ip_on_direct.patch
- [20] Henrik Nordström's Squid work
<http://squid.sourceforge.net/hno/>
- [21] RADB/Merit IRR Services
<http://www.irrd.net/>
- [22] IANA assigned numbers - Port Numbers
<ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers>
- [23] Butenhof, David R.
Programming with POSIX Threads
Addison-Wesley, 1997.
- [24] Message Passing Interface (MPI) Forum Home Page
<http://www.mpi-forum.org/>
- [25] The Globus Project: MPI
<http://www.globus.org/mpi/>
- [26] Extended Cache Statistics
<http://www.cache.dfn.de/DFN-Cache/Development/Seafood/>
- [27] IEEE computer magazine, Aug. 1999
<http://www.computer.org/computer/co1999/r8toc.htm>

- [28] Knuth, Donald E.
The Art Of Computer Programming, Vol. 3, Searching and Sorting
Addison-Wesley, 1973.

8 Further Information

Jens-S. Vöckler

Institute for Computer Networks and Distributed Systems (RVS)
University of Hanover
Schloßwender Str. 5
D-30159 Hanover
Germany

Tel: +49 511 762 4726
Fax: +49 511 762 3003
E-mail: voeckler@rvs.uni-hannover.de

Appendix A: Software

The Seafood software suite and its documentation can be obtained from

[26] <http://www.cache.dfn.de/DFN-Cache/Development/Seafood/>

The database scripts from section 4 can be found in the `dbase` folder. The web interface scripts can be found in the `public_html` folder. Further information and pointers to documentation and binaries can also be obtained at the project home page [26].

Appendix B: Deviations from the Specification

Some deviations from the original specification were introduced during the implementation. Furthermore, some approaches taken during design and coding were not as fortunate as they could have been. Finally, some minor shortcomings and possible extensions are noted in this section.

B.1 Specific Deviations

The following items were changed, omitted, or modified as compared with the system analysis presented in the above document.

- The daily interval I_1 can be configured, but is at the moment utterly unused. It is assumed that the log file(s) fed into Seafood are just about matching the configured interval. If fed with a larger interval, Seafood will not split the results into several files.
- So far, the analyser is known to parse Squid-2.2s4 log files. Some time was spent looking into 2.1p2, 2.0p2, 1.1.22 and 1.1.20 log files. This may not be perfect, yet.

- The peaks do not show any *miss* and *none* values for TCP.
- From practical judgement, e.g. seeing over 27,000 distinct 2nd-level domains per day and log file, it seems unfeasible to include the 2nd-level domain output into any kind of database. Thus, it is suggested that the analyser may print the 2nd-level domains at the administrator's request, since they are easily extracted, but due to the amount of data they will never be put into any kind of database.
- The (first) rough overview of the server side forwarded requests containing a hit count too. Usually, one should not see any hits for server side requests. Since the configuration file can be misconfigured, this is a kind of fail-check.
- The requirement for pure hits looks as if it has been met implicitly. Since the analyser can be misconfigured to count strange lines as a hit, a pure hit count would still be nice.

B.2 Thoughts on the Current Implementation

Some issues presented above are sub-optimal.

- Originally the intention was to support any database, as long as the database is able to understand ANSI-SQL. In the meantime, without deeper knowledge of what is defined in ANSI about SQL, and what is not, it is deemed feasible to target industrial grade databases like Informix, Oracle, Adabas and the like. Those database either already provide, or will in the near future provide special capabilities for handling data warehouses.
- The database design could still be improved by eliminating even more tables and creating more meta tables which save part of the configuration and known key words. Also, the database design should focus on an ANSI-SQL compatible abstract database, and leave the missing features to work-arounds in the implementation for a concrete database.
- Any extension to the prototypical interface should note that the client tables, AS table and 2nd-level domain table only contain the top-N by request items. Further items are summed up in the *more* pseudo entry. A web user interface selection may now become critical: if the user selects the volume output, some of the real high-volume destinations may be hidden in the *more* pseudo entry, just because they happen to have a low request count.
- A less fixed web interface would allow for more user interventions, e.g. when selecting the set of frequent items to show. It is thinkable that the web interface may work in several steps, like the step-by-step ordering from Amazon, starting from the common selections like interval, cache hosts and output format to special selections only valid for a chosen action.
- The database feeder works by using Perl DBC. A generic approach can only use Perl or Java as programming languages dealing with the task. The use of embedded SQL in C++ is not feasible as some vendors do not supply pre-processors, other vendors are only capable of parsing ANSI-C and the overall product is very tightly coupled to the database product. Still, it is assumed that using Java should yield an even more portable result though this is *not* part of the current project.

- It is a problem for the interpretation of data that some top-level domains use a small amount of meaningful 2nd-level domains, e.g. "ac.uk" or "com.tw" while others proliferate on the 2nd-level domain level.
- The time format can alternatively be printed as a bit rate at the request of the administrator.
- Only Oracle examples are provided. For the sake of general acceptance, some other database vendors should be included, too. Informix, ANSI-SQL and ODBM-SQL scripts are also supplied, but not tested. PostGreSQL seems feasible, too. MS Access *cannot* be used due to its limited data range - it does not seem to support 64-bit integers or some comparable data type.
- Not all tags provided by the Seafood database output are really put into the database. At the moment, the output for the request distribution is excluded from being stored into the database.
- Currently, only a top-N implementation is used for five tables. Alternatively, other approaches like a chosen set or a percentage threshold are noted, but not part of the project.

B.3 Minor Shortcomings

This section deals with those shortcomings perceived by the author and the project team. If you perceive more shortcomings, which you deem necessary to document, please contact the author.

B.3.1 What Is a HIT, What Is a MISS, What Is the Rest?

Currently, only those items tagged with the token *hit* in the configuration file are counted as a HIT. There exist *miss* and *none* tokens, but those are not used. Instead, the following algorithm is employed:

```

if ( status is UDP ) {
    if ( status is HIT ) {
        count as UDP HIT
    } else {
        // assume MISS
        count as UDP MISS
    }
} else {
    // assume TCP, though warn if not
    if ( status is TCP and HIT ) {
        count as TCP HIT
    } elsif ( hierarchy is NONE or missing ) {
        // assume TCP NONE/ERR
        count as TCP NONE/ERR
    } else {
        // assume TCP MISS
        count as TCP MISS
    }
    if ( hierarchy not NONE ) {
        count server side stuff
    }
}

```

This kind of hit counting might not meet all requirements, e.g. of log files from different vendors. The places marked with C++ comments state a kind of precondition that is assumed at that particular point. Breaking these preconditions should make the results more accurate. Any suggestions for an algorithm less prone to counting the wrong things at the wrong place are welcome.

B.3.2 Meta Traffic

Any meta data and inter cache communication exchange should be displayed separately in order to give an administrator an idea of the percentage of traffic generated just for maintaining a cache mesh. Of course, the traffic is not worthless, and some of the traffic would even be generated without caches.

Squid internal object meta traffic

There is at the moment no separate chart about the amount of meta information generated by Squid as a source of non-cache hits. It is possible to get the amount of ICP inter-cache-communication from the UDP charts, the amount of cache digest traffic from the `application/cache-digest` media type and the cache manager data from the `cache_object` scheme.

Not directly visible are other Squid generated objects like the FTP icons, or almost anything else starting out in their URL path with `/squid-internal` (`static|dynamic|periodic`). It looks feasible to generate a different chart where all the meta data is put into relation to the total data transferred. The actual instance might need to be configurable, as different vendors might use different paths.

Peer traffic

The analyser shows what amount of data was transferred from peers, but it only shows, with a short line in the client side table, the amount of data requested by peers. The client side table should be split into regular siblings like dependent caches or browsers, and peers. It is possible, and within the amount of data gathered, to make this distinction without having to parse the log file a second time. Since peering relationships need not be symmetrical, an automatic configuration will not be perfect.

If you are planning to use the analyser on many log files from the same cache mesh simultaneously, some traffic will be accounted for multiple times. Splitting the client side traffic into peers and non-peers helps to focus on the real traffic.

Summary

A summary over the previously mentioned topics should yield some insight into how much traffic was used just for maintaining the caches. The TCP traffic of hierarchy *none* might contain most of this traffic, but it also contains errors.

B.3.3 Gaps in the log file

Currently, the analyser does warn about gaps in the log file, e.g. if a cache was down. It does not (yet) warn about log files being too small or large in the sense that the log file rotation did not work. Sometimes, feeding such log files is intentional, but a configurable warning would not hurt.

B.3.4 Configurability

The configuration file is quite large at the moment, and further growth is expected to accommodate future options. Also, some choices are not really fortunate and would need to be improved. For instance, the separate warn option for elements not listed in hierarchy, status, scheme, method or TLD might be better kept as a parameter to the list, e.g.

```
status_list true { ... };
```

Talking about configurability, the libz and bz2lib functions should be compile time options in the Makefile.

B.3.5 Look up suppression

In some instances, a form of output format is needed which may or may not need DNS look ups, depending on the input file format of the `access.log` file. Currently, it is assumed that the client lists should show the host name in symbolic format, not as dotted quads. If you configured your Squid with the (not recommended) `log_fqdn` option, the log file will already contain symbolic names, and no reverse look ups on the client addresses would be necessary.

However, sometimes either an administrator would like to see the dotted quad addresses, or would like to anonymize log output by grouping by netmasks. Both options are not yet implemented. In this case, if the Squid had the (not recommended) `log_fqdn` option activated, forward look ups will be necessary.

Basically, four options spring into mind when pondering the perceived problem of printing client addresses. Similar thoughts apply to the destination address in the AS listing.

1. The client addresses should always be printed in symbolic form, regardless of the form used by Squid.
2. The client addresses should always be printed in dotted quad form, regardless of the form used by Squid.
3. Client addresses should be combined by a configurable netmask. Since network symbolic names are rarely configured in the DNS database, this will result in the dotted quad form.
4. The client address should be printed in the same form Squid uses, so that no look up whatever is used.

B.3.6 URL Parsing

The finite state automaton described in section 3.5 has one obvious weakness. It is unable to correctly determine the scheme, host name or port, if any of these items contain URL escaped characters. For the time being, it is assumed that none of these three items contain escaped characters.

Also, well-known symbolic port names are not really understood. Though RFC 2396 (URI syntax - [9]) claims that port numbers in URLs should be given as digit string, some IANA defined well-known ports [22] are understood by almost all cache hosts

operating systems. Still, Squid does not understand about symbolic port names, either, therefore all ports are logged numerically.

B.3.7 Multi-processing

Threads are planned for obtaining an even higher throughput on multi-processor machines. The decision on the correct granularity to use is a difficult one. It is believed that giving log file readers a thread of their own will improve compressed input speeds, for a start. Also, creating disjunct sets of counters and feeding them to a work line might give some more throughput. Using a work crew for these parts [23] might further speed-up processing.

On the other hand, looking at heterogeneous computer pools in universities, or just multi-processor nodes without shared memory, the MPI toolkit [24] [25] looks more feasible for inter-process-communication in such environments. On multi-node shared-memory machines, it imposes an overhead, but the approach is currently regarded as more flexible than threads.

Anyway, with almost 40,000 lines per seconds and the increasing tendency for more powerful platforms, parsing a days worth of log files is a matter of minutes, and thus multi-node processing is not really as urgent for the parsing process.

Threads come in handy when parsing compressed log files. Giving the decompressor a thread of its own should speed up performance on any multi-processor machine. Currently, whenever a multi-CPU machine is detected and more than one CPU is online, Seafood tries to start the decompressor in a separate process. Thus some of the speed of multi-processor machines will be handed to you.

Appendix C: Implementation Details

This appendix deals with some gory details of the implementation that were hinted at in section 3.6. Working with the standard template library (STL) [10] of C++ introduced an incredible amount of portability problems. Many compilers still lack important features in order to achieve a common level of conformance. At the time of writing, it does not seem possible to write programs that are *portable* between many platforms and compilers while still employing STL. For this reason, the usage of STL constructs in Seafood was kept to a minimum. Many areas where Seafood might have benefited from STL were coded using standard techniques.

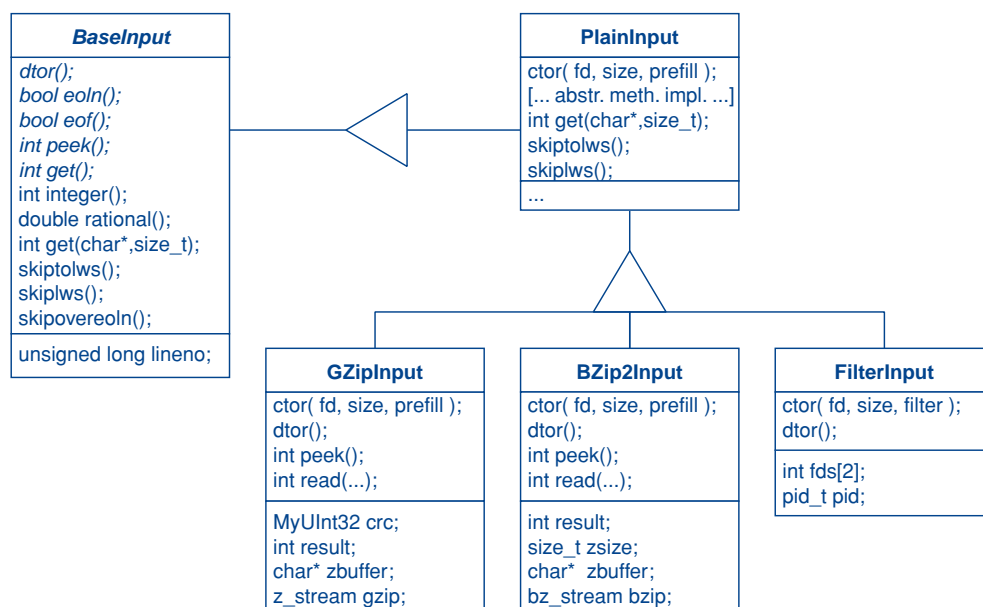
C.1 Reading Log Files

Figure C.1 shows the class hierarchy of the log file reader. The design is neither good nor beautiful, but it is efficient.

All access to any input method will be done via a base class pointer. By this concept, arbitrarily compressed or encrypted log files may be added to the analyser. All that is needed is another class being able to parse the new input. The compression classes are siblings of the plain text class for efficiency and code reuse reasons. Details can be found in the source files `input.*`.

Part of the current implementation is the BaseInput hierarchy, shown in figure C.1. The abstract base class defines the outer visible interface. The siblings implement the interface. The primitives shown are sufficient for parsing the Squid logs, but as other log formats emerge, the class will certainly need extensions. The base class also contains a few implementations, which all rely on the `peek()` and `get()` method. Those are central methods to the input.

Figure C.1:
Class
hierarchy
for log file
input



`peek()` returns the look-ahead character, and includes the states error as -1 and end-of-file (EOF) as -2. `get()` obtains the character by removing it from the buffer. The return values follow the `peek()` semantic. All input buffer handling in sibling classes should be managed in `peek()`. `get()` is more like a `peek()` call with a subsequent advance cursor call.

The higher level parsing functions `integer()`, `rational()` and `get(string)` rely on the lower level functions. The integer functions come in four flavours, able to parse 64-bit and 32-bit signed and unsigned numbers. Effort was taken to supply an efficient implementation, copying buffers as the cursor is advanced. No character sequence should need to be read twice. Thus, the input performance is often slightly better than standard IO performance and a lot better than C++ streams.

The `PlainInput` class implements the interface functions. It provides fully buffered input for the supplied file descriptor. It also overwrites almost all base class methods with a more efficient refinement.

`GZipInput` is logically a sibling of `BaseInput`, and was planned as a brother to `PlainInput`. Implementation though showed that it was easier to implement as a refinement of `GZipInput`, thus using the more efficient implementations and only modifying the `peek()` method to work with zipped files. Similar observations are true for the `BZip2Input` class.

The `FilterInput` class starts an external program, connects the given file descriptor to the `stdin` of the external filter, and connects the `stdout` of the external filter to the file descriptor the `PlainInput` parent class reads from. With the help of the filtered input, on multi-processor machines the decompression can be sped up while maintaining the easy to use command line interface.

The overall performance when using the currently non-threaded internal decompressor on a symmetric multiprocessing computers is worse, when compared to an external unzip piped into Seafood, or the filtered input processor (which runs in a separate process).

On a single CPU system with a decent scheduler the internal unzip mechanism performs as good as the externally filtered input. However, putting the input classes into threads of their own should always be considered beneficial.

C.2 A Trie Matcher

The central data structure enabling the analyser to its perceived speed is the trie data structure. The word originally derived from retrieval, but in order to distinguish it from a regular tree, it is often pronounced like try-ee. A trie is a non-binary tree, where each node contains one letter of the word to be matched. With each letter, one level in the tree is descended. Common prefixes are thus bundled together.

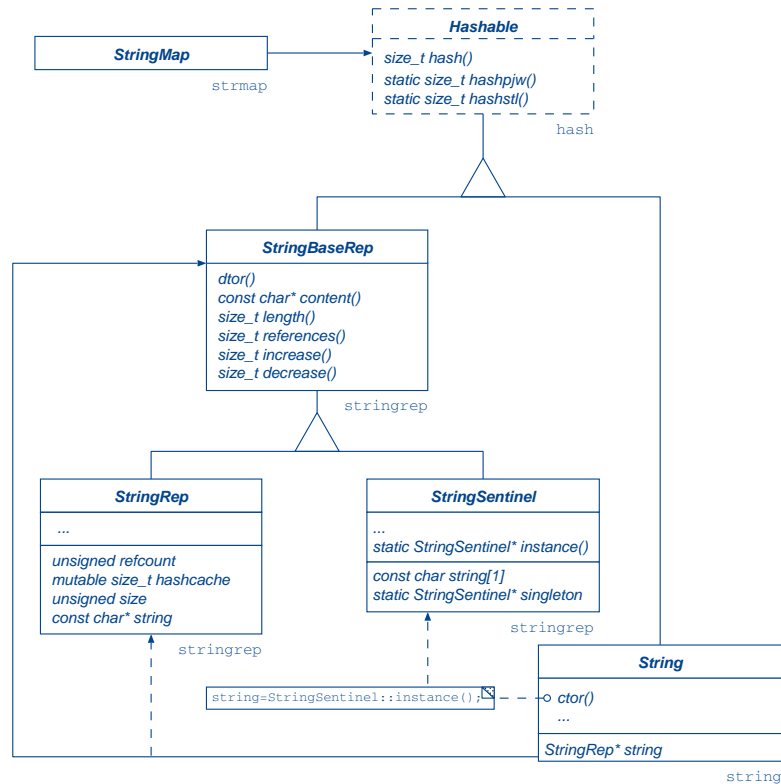
There are currently four different implementations of a trie, three of which are actually used for the project:

1. One implementation limits the words matchable to all uppercase letters and an underscore. The implementation is vector based, and each node can have a maximum of 32 children.
2. There is a similar implementation limiting matchable words to ASCII characters. Since this second implementation also uses a vector, this time of 128 siblings, it is quite memory intensive, and should only be used for short words and word lists with lots of common prefixes.
3. Another implementation uses linked lists to be more memory efficient, and with the knowledge that in the lower level of the trie, often there is only one sibling. Of course, searching this kind of trie is slower.
4. There is an unused implementation of a trie which tries to be both memory efficient, unlimited and fast. It uses an expandable vector approach, and a character table indirection. Unfortunately, it does not work (yet), and was thus excluded from the project.

C.3 The String Implementation

In order to supply Seafood with similar high level constructs as Perl, namely strings and hash tables, appropriate data types were created. The string class family is primarily used for indexing associative arrays and returning symbolic information from functions. Figure C.2 sketches the interrelation of the different classes handling symbolic information.

Figure C.2:
Chart of
string class
interrelations



All maps are indexed by a `String` and may use an arbitrary value class. Only basic C types need to set the Boolean argument to the map, indicating that there is no default constructor setting the correct start value during vector construction.

The string map only uses the `String` type as a key, but additionally enforces that its keys conform to the `Hashable` interface. The term *interface* was taken from Java, but appropriately describes the function of the class. As the class is abstract, thus shown in italics, a sibling must supply the abstract methods, if it does not want to be abstract itself. The `Hashable` interface is common, and thus needs some refinement with regards to strings.

The base class `StringBaseRep` is also an abstract interface, but provides a refinement of `Hashable`. Of its two sibling classes `StringRep` and `StringSentinel`, only the former is used for storing "real" strings. The sole purpose of the sentinel class is to speed up the default constructor of `String`. The special properties of the sentinel are that it always returns an empty C string as content, has the length 0, has `MAX_INT` references, and the increment and decrement arithmetics have no effect upon it. `StringSentinel` is realized as a *Singleton* design pattern. There will be only one *instance* of the class, and access to it is granted through the `instance()` method. As `String` contains a base class pointer

to `StringBaseRep`, and `StringSentinel` is a valid `StringBaseRep`, the default constructor of `String` just stores the address obtained through the singleton. Please mind that the default constructor is also called for the construction of `String` arrays, as this is where the performance gain lies.

Class `StringRep` on the other hand goes through a real `new` and `delete` calls for constructor and destructor respectively. Any other `String` constructor than the default constructor will really create a `StringRep` object. The reference counter in `StringRep` and the outward interface provided by `String` handle things like copy-construction and assignments in an efficient manner by doing arithmetics on the reference counter. Working with reference counters creates shallow copies. You can create a real deep copy by invoking the `clone()` method, but that has not been necessary in this project. It has been said that the reference counting ability will prove harmful to multi-threading the code!

Talking in design patterns, the relation between `String` and the representation hierarchy is in one way like the *Bridge* pattern. The `string` class defines the outer visible interface, whereas the representation classes implement string storage. Also, the relation behaves like a *Flyweight* in another way, because many objects can be shared efficiently.

