TERENA TF-Cache
Small project 99-005
Extended Cache Statistics

Seafood - a log file analyser (Deliverable 2, Update 3)

Jens-S. Vöckler

3. September 1999

Table of contents

The DFN caching project is sponsored by the German Ministry of Education, Science, Research and Technology (BMBF) through the German Research Network Association (DFN Verein). The Trans-European Research and Education Network Association (TERENA) sponsors the "Extended Cache Statistics" project.

1 Compiling Seafood

The seafood log file analyser is part of a TERENA sponsored project[1]. Please note that this is an early version of the parser. It is known to parse Squid-2 access.log files. Older versions of Squid might be supported, but that was not tested. Those might need some changes to the configuration file. But I am getting ahead of myself here.

If you are familiar with calamaris[2], you will recognize the output format.

In order to successfully achieve a compilation of the seafood software, you will need either a g++ 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[3] constructs. Egcs might do the trick, too.

If you don't have such a compiler, you will need to change some things in the `Makefile`, see the comments therein. If your system is not one of those mentioned in table 1, you might need to change some things in the Makefile, too. AIX isn't supported, because our compiler is too old and doesn't have all the features mentioned earlier. AIX might work with a GNU compiler. Some compilers use an obsolete or no version of the STL, but might be brought up to date using the STLport[4].

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

Table 1: Systems and compilers known to work.

Seafood was successfully translated on the systems mentioned above, though with Irix, only the native compiler will do the trick. The later GNU compilers should be able to compile the project on many systems not specifically mentioned above.

1.1 Installing compression libraries

The seafood log file processor can read compressed files. Compression is an option you can turn off during compile time in the `Makefile`. Just leave the `ADD_FLAGS` macro empty of those libraries you don't have installed, and comment out any material related to your particular compression library in section 1 of the `Makefile`.

Seafood is 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, refer to section 5.7 (page 21).

---

[1] http://www.cache.dfn.de/DFN-Cache/Development/Seafood/
[2] http://calamaris.cord.de/
[3] http://www.sgi.com/Technology/STL/
[4] http://www.stlport.org/

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

1.1.1 libz

Refer to the zlib homepage[1] on where to get your version of libz. Unpack it, follow the instructions, and install the library `libz.a` into `/usr/local/lib` and the header files `zconf.h` and `zlib.h` into `/usr/local/include`. Some systems will need a `ranlib` call, if you move a library.

If you cannot install into `/usr/local`, you can use another directory, but you will need to change the `ZLIBDIR`, `ZLIBINC` and `ZLIBLIB` macros in the `Makefile` to reflect the correct location, e.g. most Linux systems have it installed by default, not needing to have any of these parameters set.

Set the Makefile parameter `ADD_FLAGS` with the GNU make append assignment operator `+=` to the value `-DUSE_LIBZ`. By supplying the flag, the relevant source files are informed to include the appropriate headers and pieces of source code. If you do not want to use zlib, comment out `ADD_FLAGS` and the `ZLIB*` flags.

By default, the use of the libz compression library is enabled in the `Makefile`.

1.1.2 libbz2

Refer to the bzip2 homepage[2] on where to get your version of bzip2. Unpack it, follow the instructions, and install the library `libbz2.a` into `/usr/local/lib` and the header file `bzlib.h` into `/usr/local/include`. Some systems will need a `ranlib` call, if you move a library.

If you cannot install into `/usr/local`, you can use another directory, but you will need to change the `BZDIR`, `BZINC` and `BZLIB` macros in the `Makefile` to reflect the correct location, e.g. most Linux systems have it installed by default, not needing to have any of these parameters set.

Set the Makefile parameter `ADD_FLAGS` with the GNU make append assignment operator `+=` to the value `-DUSE_LIBBZ2`. By supplying the flag, the relevant source files are informed to include the appropriate headers and pieces of source code. If you do not want to include bzlib, comment out `ADD_FLAGS` and the `BZ*` flags.

By default, the use of the bzlib compression library is disabled in the `Makefile`.

1.2 GNU make, flex and bison

The source files are checked out read-only to user `voeckler`, sorry for the inconvenience. If you have RCS installed, you may check it writably on your account.

Since I don’t understand the GNU autoconf process, and didn’t have time to dig into that, the Makefile will try to distinguish between different systems, therefore I need the extended GNU flavour of make. With a regular make, you are bound to run into problems.

---

[1] http://www.cdrom.com/pub/infozip/zlib/

[2] http://www.bzip2.org/

The `Makefile` contains some comments which are meant to aid you. It is split into three sections. The initial section contains tries to figure out the System you are using and set the necessary parameters, like the above mentioned compression libraries. The second section is made up of rules for different host systems and native compilers. The final section should not need changes, and basically states the dependencies.

There are some compile time configurable parameters for those systems not mentioned in table 1 (page 2):

- You should use the `-DHAS_BOOL` define, if your compiler known about the data type `bool`. If the flag is not set, a work-around will be used.
- If your compiler knows about the key word `mutable`, the `-DHAS_MUTABLE` flag should be turned on.
- If your C library supports the non standard character class predicate macro `isblank()`, as the GNU libc does, you should set the `-DHAS_ISBLANK` flag.
- Finally, some systems are notoriously slow when using the character class predicate macros, because they do a wide character checking and sometimes check back with the locale. If the `-DCTYPE_IS_FAST` flag is **not** set, this kind of performance eating behavior is avoided. On the other hand, if the test program `chkctype` finds that your system library is sufficiently fast[1]:

  ```
  $ ./chkctype /boot/vmlinuz # use any large file
  REGULAR: 439916 bytes, 16665 digits, 10171 spaces, 10.791 ms
  NEW VER: 439916 bytes, 16665 digits, 10171 spaces, 9.814 ms
  ```

  even if it is a little slower than the new version, you should safely set the `-DCTYPE_IS_FAST` flag. That way, the libc macros will be used.

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, though I might try in the future.

1.3 The actual compilation

Just say make and watch the compiler go to work. Don't become exasperated with the impossible amount of compile time needed for all the files which contain STL material. That is perfectly normal. With Linux, Solaris and Irix, some optimizations are supplied. Some parts of the code can cause an older version of g++ to break, thus for the counters.cc module all optimization is turned low explicitly - it won't hurt that particular piece of code anyway. Other (few) pieces can safely be compiled with maximum optimization. Some compilers are notorious for breaking code with too high an optimization.

Just a note, if you are using a Linux, BSD, or Solaris x86 on a Pentium II or above, and a rather recent gnuish compiler, the compiler option `-mcpu=pentiumpro` will really make things still a little faster. You might also want to risk a sneak preview into some of my host specific configurations for special compiler tunings, e.g. origin is an Origin-200, blau is an UltraSPARC-2 and grafzahl a Pentium-II.

---

[1] `chktype` does not sufficiently test your system. The dependency on your current locale settings are ignored in this test program, too. Using a non-C locale may slow things down.

Expect a few warning from your compiler about unused `RCS_ID` variables or some statements not reached in the configuration parser.

1.4 Installation

If you are planning on parsing Squid-2 log files, you won't need much fiddling with seafood.conf. Still, 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[1]. If you are located in Europe, you might wasn to try the RIPE whois server. If your access is from within an European research network, you might want to try `whois.rvs.uni-hannover.de`. Note that it only offers a very limited capacity.

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.

---

[1] The constraint of having to use a RA compatible whois server was removed.

2 Configuration options

This section describes the options which can be used in the `seafood.conf` file. Please note that the configuration file has to reside in the same directory as the binary, and needs the same base name. An accurate and up-to-date description of the configuration options can be found within the comments in the log file.

option	**arguments**	**defaults**
no_ident	<bool>	false
debug_level	<int> [, <int>]	0,0
prefer_datarate	<bool>	false
peak_interval	<int>	3600
daily_interval	<int>	86400
warn_crash_interval	<int>	1800
show_distribution	<bool>	false
log_fqdn	<bool>	true
dns_cache_file	<string>	"/var/tmp/dns"
dns_positive_ttl	<int>	2592000
dns_negative_ttl	<int>	604800
dns_program	<string>	"dnshelper"
dns_children	<int>	16
dns_length_tlds	<int>	0 (unlimited)
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>	section 2.3.2
warn_unknown_method	<bool>	false
hierarchy_list	<id list>	section 2.3.3
warn_unknown_hierarchy	<bool>	false
status_list	<id list>	section 2.3.4
warn_unknown_status	<bool>	false
scheme_list	<id list>	section 2.3.5
warn_unknown_scheme	<bool>	false

Table 2: Syntax of all currently known configuration options.

option	arguments	defaults
mediatype_list	<string list>	section 2.3.6
media_subtype	<string> <string list>	section 2.3.7
domain_list	<string list>	section 2.3.8
warn_unknown_tld	<bool>	false

Table 2: Syntax of all currently known configuration options.

Please mind that this is an early release. The amount of configuration options most certainly will change in the future, the options themselves might change their syntax, too, or get renamed.

The grammar of the configuration file is (almost) format free and case sensitive. Whitespaces and empty lines are ignored. Comments are started with a hash # sign, and extend to the end of the line. All configuration options have to be terminated with a semicolon - this is more like Pascal than like C.

Table 2 shows the different options at a glance. The `<bool>` type may either take the value `true` or `false`. 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.

An `<id list>` is a list of unquoted words which may only contain upper case letters and underscores. No digit are allowed. The lists items may be modified by special token words. Not all modifiers make sense for all lists. Each list item is separated by comma from the next. The list itself is enclosed in curly braces.

The `<string list>` is similar to the id list, though each element is a quoted string, and none of the item modifiers are allowed.

For all configurable values, reasonable defaults are assumed. Still, the configuration file must at least be touched into existence.

2.1 Flag items

The `<bool>` type configuration options may either take the value `true` or `false`, regardless of the case it is written in.

2.1.1 no_ident

If you disabled the ident protocol logging in your file - see the "ident_look up off" option in your squid.conf - the analyser will be enabled to correctly recognize URLs containing whitespaces.

2.1.2 prefer_datarate

Display a data rate in **bit** per second instead of a duration. If you chose the duration, time will be scaled in milliseconds, seconds, hours, days and siderian years.

2.1.3 log_fqdn

Turn on this option if you wish to log fully qualified domain names for the client side lists. To do this seafood does a DNS look up of all client IP addresses. Such an action will increase latency. If your squid uses the **depracated** `log_fqdn` option, (almost) no DNS reverse look ups need to be done for the client side.

2.1.4 show_distributions

Use this option to toggle the display of the size-time, size, and time distribution of TCP based HIT, MISS and NONE requests. Those tables are rather broad.

2.1.5 warn_unknown_XYZ

The warn options, if turned on, protocol all unknown list item for list XYZ as informational warning onto `stderr`. As you can see, the warnings are a kind of debug option when new status codes, hierarchy tags, HTTP methods, request schemes, or top level domains were introduced. For the latter two items, the warn option is not too meaningful.

2.2 Interval options

The `<int>` type allows for non-negative integer numbers. Hexadecimal, decimal and octal constants are recognized, if the C prefixing is used. All intervals are specified using seconds. The look up services intervals are explained in section 2.4 (page 10).

2.2.1 peak_interval

The amount in seconds for which a peak value will be accumulated. The so-called peak data should really be called performance data.

2.2.2 daily_interval

Currently work-in-progress. This is the amount of time for which data will be accumulated. After the interval is reached, the accumulated data will be dumped into the data base, and a new set of counters will be started to gather further information.

2.2.3 warn_crash_interval

The amount in seconds between to consequetive time stamps in the log file when a warning about a possible cache crash should be displayed. This is in effect a ’silence time’, when the service, host, or network may have been down, or simply nobody asked your cache. Thus you need to set it sufficiently large not to be bothered with wrong warnings, but sufficiently low to detect down times. The actual value depends on your cache and clients.

2.3 List items

2.3.1 List item aliasing

Both list types allow aliasing. That is, you can give an alternative name for a previously configured option. The concept is similar to a symbolic link:

```
<new value> alias <old value>
```

The new value is the new name. If the list requires quoted items, both values most be set in double quotes. Aliases can only be used within the same list environment. The old value referred to must already be defined previously in the list. Modifiers are not permitted with an alias. The aliased new names are never part of the output. Only the referenced old name is part of the output.

2.3.2 The method_list

The method list is by default empty, but seafood.conf file contains all methods known to HTTP/1.1 [RFC 2616], Squid-2 (NONE, ICPQUERY, and PURGE) and those extensions used by WebDAV. You should use the methods from the supplied configuration file. There are no modifiers or aliases used.

2.3.3 The hierarchy_list

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 should 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. You can configure the appropriate list item to suit your needs.

2.3.4 The status_list

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 should have exactly 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.

2.3.5 The scheme_list

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.

2.3.6 The mediatype_list

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 (well, the author of web pages and server side includes), you would see many strange things here. For the reason of media types being user input without Squid sanity checks, a warning options was not deemed feasible (yet).

2.3.7 The media_subtype lists

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>`.

2.3.8 The domain_list

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 number.

The virtual top level domain `<empty>` represents malformed URLs. The `<unknown>` value will be used for anything not in the list, and also represents malformed URLs, though of a different malfunction. The interested user may refer to section section 5.1 (page 19).

2.4 External look up services

Give special attention to the values of the options `log_fqdn` and `irr_server`. 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 just a small amount of time.

The option `irr_server` is very needy in time consumption. If you are using AS based look ups, you should have applied the log_ip_on_direct patch. 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. Refer to section 3.2.8 (page 15) for details on how whois look ups affect seafood.

2.4.1 dns_cache_file

The cache file will be read into main memory after the configuration file was parsed. It resides in main memory until seafood terminates regularly. At finalization, the contents of the cache will be dumped into the given file. Please have enough space available, as the DNS file can grow quite large when used in conjunction with the the AS look ups.

2.4.2 dns_positive_ttl and dns_negative_ttl

The `dns_positive_ttl` option specified number of seconds a positive (successful) DNS look up should stay in our local DNS cache before it will be verified. The negative TTL specifies the amount of time, the cache should not try to re-validate unsuccessful look ups.

2.4.3 dns_children and irr_children

The number of external helper processes that should be spawned. The number is limited between 1 (minimum) and 255 (maximum). Your operating system limits may decrease the lower limit, e.g. if the number of open files or the number of processes per user is further limited. Seafood will try to determine these threshold, and allows at maximum only half of what the OS claimed to allow you.

Also note that most whois service providers do not allow a misuse of their service, thus set the number of IRR children sufficiently low.

2.4.4 dns_program and irr_program

The fully qualified path name of the external helper program called `dnshelper` or `irrhelper`. If you are starting seafood from a script, you should set this parameter to the correct value. The default is a path name relative to the current working directory.

2.4.5 irr_server

This is the name of the whois server which is willing to service the requestion. Please note that the restraint of using a RA or MERIT compatible daemon was removed. You can now safely use any complete whois mirror near you. The drawback is that persistent connections are no longer used for whois look ups.

If you want to disable the AS feature, use the unquoted word `none` as server name. Also note that most whois service providers do not allow a misuse of their service. If you have extensive need of the service, set up a mirror of your own.

2.4.6 irr_cache_file

This is the name of the text file to cache IRR look ups and IRR reverse look ups. Use somewhere inconspicious where you are sure to write and grow.

2.4.7 irr_positive_ttl and irr_negative_ttl

The same as the DNS intervals, though for the IRR look ups. The `irr_positive_ttl` option specified number of seconds a positive (successful) whois look up should stay in our local IRR cache before it will be verified. The negative TTL specifies the amount of time the cache should not try to re-validate unsuccessful look ups.

2.5 Tables with configurable lengths

Some of the tables have an abundant or nearly unlimited amount of data. Those tables can be limited in their lengths. The default is not to limit any table, signaled by a value of 0.

2.5.1 dns_length_tlds and dns_length_slds

This set of length configuration limits the top level domains and 2nd level domains table respectively. The top level domains table is also limited by the number of explicitly named and configured top level domains, see section 2.3.8 (page 9). 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 30000.

2.5.2 client_length_udp and client_length_tcp

These items configure the number of clients using your cache. There are two tables, one for the UDP clients and one for TCP clients. In a multi-level caching hierarchy, the top level caches may only see a small number of sibling caches, but local university caches usually see all browsers from all computers in the network. Therefore, a limitation of the client tables seems feasible.

2.5.3 irr_length_asns

This parameter limits the size of the AS number look up table. The value will only be taken into account, if the generation of an AS table was required by the `irr_server` option.

3 Running Seafood

Run seafood on one or many of your log files. Each run will summarize all log files poured into it. Each log files may be plain, gzip compressed or bzip2 compressed, at your option. `Stdin` can be used by the special file name dash (`-`).

Again, before running seafood, check your configuration file, and make really sure that you configured those options you want. Give special attention to section 2.4 (page 10) for a speedy printing of the results. If you switched from an earlier version, check with the CHANGELOG file, if you need to remove the DNS and IRR caches.

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

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

3.1 Stderr

On `stderr`, a bunch of 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.

```
# machine is big endian
```

The endianess detection of the host machine is done during run-time, because there is no POSIX way to do it with portable header inclusions during compile-time. It is only of relevance to the network and netmask matching code. Machines other than little- or big endian (e.g. PDP) result in an error, as the required CIDR conversions are not yet coded.

```
# maxfd hard=2500, soft=200, sys=200
# nproc sys=522
```

These information pertain to the number of usable external helper processes. The maxfd row shows the hard- and softlimit as well as the system information of the limit. The nproc row shows the number of user processes. SGI does not have a resource limit, so you won't see a number for the hard- and softlimit. 45 % of the smallest of these numbers is the maximum number of external helper processes of one kind you can start.

```
# trying to read "./seafood.conf"
# ...10 ...20 ...24 ...34 ...42 ...48 ...53 ...58 ...66 ...72 ...77
[...]
# ...606(250) ...611
# done reading "./seafood.conf"
```

This is the informational part of reading the config file. Whenever a complete statement in the configuration file was parsed, its line number will be displayed. With list statements, the number of list members will be put into parenthesis.

```
# trying to read DNS cache and start DNS co processes...
# DNS cache contains 6278 entries.
```

If you configured seafood to to host name look ups, it will try to read its DNS host name cache from a previous run. The data comes from the `/var/tmp/dns` text file. Also during this point of time, the DNS helper children, external processes in the Squid style, are started.

```
# read IRR cache, start IRR co processes, use server "whois.ripe.net".
# IRR cache contains 1799 entries.
```

If you configured seafood to provide the AS number based table, the external whois look up helper processes are started, using the server indicated in your configuration. Also, the textual cache file `/var/tmp/irr` from a previous run is read.

```
# using external filter "/opt/gnu/bin/gunzip"
```

If you are using a compressed file to be processed, depending on the number of CPUs in your machine an external decompressor will be started. All your `PATH` will be searched for the known decompressors `uncompress`, `gunzip` and `bunzip2`, depending on the magic of the file. If you only have one CPU, the internal decompression functions will be used[1]. The message above is from a two CPU machine.

```
# 65536 lines processed from this file
# 131072 lines processed from this file
# 196608 lines processed from this file
# 200000 lines processed from this file
# done processing files, writing results
```

These informational messages are a kind of progress indicator. They may be interrupted with actual warning like the following:

```
7: unknown URL scheme ""
```

which means that on line 7 in the input file, a URL without a scheme was supplied. In this case it was a `CONNECT some.host:443`, so indeed there was no scheme to the URL.[2]

```
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, compare with section 5.1 (page 19). It is just the informational warning, that your results may be off, though in this case counting as TCP is correct.

```
# title...
# overview...
  [...]
# peaks...
# stats...
```

After the log file was processed, the output will be generated. Each section for the output has a respective log associated.

```
# dumping DNS cache (6278 entries)
# dumping IRR cache (1799 entries)
```

If the DNS and IRR helpers were used, the internal cache contents are dumped into their respective textual database files. Using the cached entries will speed up future requests.

```
# DONE!
```

DONE signals the finish of the output.

---

1. The `uncompress` algorithm is not available as internal decompressor.
2. You can disable this warning by setting `warn_unknown_status` to `true`.

Do not be disturbed if the seafood seems to sit waiting on one of those mentioned in table 3.

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.

Table 3: Reasons for possibly slow output.

3.2 Output: writing the results

There are a number of tables written to `stdout` after a successful parsing. Due to possible look ups, the writing may take some time, too. If you are familiar with calamaris[1], you will recognize the output format.

3.2.1 Overview

The overview section contains a simple table just displaying the sums of all TCP, all UDP and SUM traffic, including hit rates.

3.2.2 Status

The status section contains a UDP table, which is sorted by HIT and MISS, and a TCP table, sorted by *hit*, *miss* and *none*. Check your configuration file and section 2.3.4 (page 9), if you encounter troubles with the status list.

3.2.3 Hierarchy

The hierarchy first 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, it for some hierarchy codes also include parents. The tag lines contain a hit rate, which is nonsense, and serves as 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. Check your configuration file and section 2.3.3 (page 9), if you encounter trouble while running seafood.

3.2.4 Methods

The hierarchy information is followed by the request method table, sorted by requests. WebDAV is part of the configuration file, and will thus be counted. Unknown methods are counted as `<unknown>`. For troubles, also check with section 2.3.2 (page 9).

---

[1] http://calamaris.cord.de/

3.2.5 Schemes

The URL scheme is part of another table, sorted by requests. Unknown schemes are counted as `<unknown>`. Check with section 2.3.5 (page 9) for more schemes.

3.2.6 Top level domains

The top level domains as configured are displayed in the following section. The first table is sorted by requests, the second one by volume. If you want to limit the output, check with section 2.5.1 (page 11).

3.2.7 2nd level domains

The 2nd level domains are displayed in the following two tables. The first table is sorted by requests, the second one by volume. Both tables can be limited in their extent, as described in section 2.5.1 (page 11).

3.2.8 Destination AS

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, sorted by requests and sorted by volume, will be displayed. Together with any information from your NOC, you will be enabled to get an idea how costly a certain part of the traffic is, and what links are likely to be hit. The virtual entry `<NOIRR>` implies that for the given host, an origin AS could not be determined. Both tables can be limited as described in section 2.5.3 (page 11).

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 mirror service can be queried. If you want to disable the AS feature, place the following server in your seafood configuration file:

```
irr_server none;
```

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. So you really might want to use either a mirror closer to you, or use your own mirror (I do!), see the instructions at the RA whois server[1] on how to install a mirror.

Note, such a whois mirror server can also be used for Squid, see `as_whois_server` in your `squid.conf`. Refer to the ACL schemes `dst_as` and `src_as`.

3.2.9 Media types

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.

3.2.10 Non standard ports

It might be of interest to know which other ports besides 80 were used. In order to reduce the output volume, the ports are grouped by 1024 ports. Port 80 is mentioned separately.

---

[1] http://www.irrd.net/

3.2.11 UDP clients by request

The UDP client side table is just sorted by requests. The table displays the UDP traffic and the part of the *hit* traffic. The table can limited as shown in section 2.5.2 (page 11)

3.2.12 TCP clients by request

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 table can be limited as shown in section 2.5.2 (page 11).

3.2.13 Cache performance

The performance data are associated with 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.

3.2.14 TCP distributions in time and size

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.

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.

My `matrix` tool has yet to be updated to parse this table. The tool creates 3D diagrams, a 3D VRML and 2D distribution diagrams from the data.

3.2.15 Statistics

The final table displays some statistics about the analyser, e.g. how much time it spend in the parser loop and how much time it needed at all. Of interest may be the lps value in parenthesis which is the lines-per-second amount parsed.

Do not be disturbed that even with empty DNS and IRR cache files, you will get hits on the DNS and IRR cache. Mind that some tables come in just two different sortings, and thus the second table will hit the cache with its queries.

4 Some questions and answers

1. My seafood complains about opening its DNS cache file, and dies?

   Probably a previous run of seafood was somehow killed or interrupted, and left the cache database(s) in an inconsistent state. Remove the cache database files, and restart seafood. This was true for the old NDBM version.

2. My seafood seems to run, but I get very many warnings?

   The current version was tested and run against log files from a Squid-2.2.STABLE4, though 2.1.PATCH2, 2.0.PATCHx, and 1.NOVM.22 were examined, too. If you are using a different version, there may be slight changes in what is logged in what manner. Examine the lines printed as warning, the warnings, and change the configuration file to include the tags from the Squid version you are using.

   Alternatively, an experienced user may look into the Squid source tree. The status tags are in `access_log.c` and the hierarchy tags in `peer_select.c`.

3. What is a HIT, what is a MISS and what is counted anyway?

   Refer to section 5.1 (page 19) and to the `status_list` configuration option in the configuration file.

4. I think I found a bug?

   Please contact the author[1].

5. I am at home, not connected, but I want to run a quick analysis?

   Use the `-N` command line option to switch off any look up.

6. The performance data columns don’t sum up to 100%?

   The missing part is from the *none* traffic, which is not part of the performance output.

7. I have a multi processor machine. Shouldn’t I pipe the output of my decompressor into seafood instead of using seafood’s internal decompressor?

   For multi processor machines, an external decompression utility is started, thus effectively giving you the same kind of performance as piping the decompressed results into `stdin`. If you intend to use `stdin`, use the reserved file name dash (`-`) for it. On some

---

[1] mailto:voeckler@rvs.uni-hannover.de

systems, you can also use the special `/dev/fd/0` file name, or a named pipe.

8. What is the file format of the DNS cache file?

   The DNS cache file is a linefeed separated textual file which is supposed to be portable between different platforms. It constists of two columns. The first colum holds the key, which is a symbolical host name or a dotted quad host address.

   The second column contains the value in parenthesis. The first value is the expiration time of the record. If the key resulted in a negative DNS look up, that is all. If not, the next value is the fully qualified domain name. Further values include all known alias names and all known addresses as dotted quad. Note that the FQDN, the aliases and the addresses are all entered as keys, too.

9. What is the format of the IRR cache file?

   The IRR cache file is similar to the DNS cache file. It also consists of two columns. The first column contains the key, either a network address as dotted quad with CIDR mask, or an AS number, starting with the letter "AS".

   The value column contains the expiration date. If the look up was positive, for networks the AS number is saved. For AS numbers, the first line of the description is saved. The format may change slightly in future releases.

5 Missing things (to do list)

This section deals with the shortcomings of version 990809. The differences and deviations from the TERENA deliverable D1 documents are shown in section A (page 29).

This section deals 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[1].

5.1 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 hit. There exist *miss* and *none* token, 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 which is assumed at that particular point. Breaking these preconditions should make the results more accurate. Any suggestions to an algorithm less prone to count the wrong things at the wrong place is welcome.

The loop contains a work-around for the old Squid bug of TCPHIT/30[12] with a hierarchy code of not *none*, which is set to *none*.

5.2 Meta Traffic

Any meta data and inter-cache-communication exchange should be displayed separately, in order to give an administrator some kind of figure to see what percentage traffic was gener-

---

1. mailto:voeckler@rvs.uni-hannover.de

ated just for maintaining a cache mesh. Of course, the traffic is not worthless, and some of the traffic would even be generated without caches.

5.2.1 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.

5.2.2 Peer traffic

The analyser shows what amount of data was transferred from peers, but it does only show 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.

5.2.3 Summary

A summary over the previously mentioned topics should yield some insight on 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.

5.3 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.

5.4 Configurability

The configuration file is quite large at the moment, and I will expect it to grow further to accommodate for future options. Also, some choices are not really fortunate, would need different constellation etc. For instance, the separate warn option for elements not listed in hierarchy, status, scheme, method or TLD might be better kept as 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. I know it is easily done, but there were more important things to tend to.

5.5 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, 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 address is necessary.

But sometimes, either an admin would like to see the dotted quad address, 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:

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.

Similar thoughts apply to the destination address in the AS listing. A recent patch[1] by Henrik Nordström's[2] allow for the direct addresses to have their socket peer address logged. As the address is logged in dotted quad format, time-consuming DNS forward look ups can be avoided.

5.6 URL parsing

The finite state automaton described in section 6.1 (page 23) has one obvious weakness. It is unable to correctly determine the scheme, host name or port, if any of those contains URL escaped characters. For the time being, it is assumed that any of those three items does not contain escaped characters.

Also, well-known symbolic port names are not really understood. Though [RFC 2396] claims that port numbers in URLs should be given as digit string, some IANA defined well-known ports are understood by almost all cache hosts. Fortunately, Squid does not understand about symbolic port names, either, so there is nothing to worry yet.

5.7 Multiprocessing

Threads are planned for obtaining an even higher throughput on multi processor machines. Back at the W3C'98 in Manchester, there was a discussion about the amount of granularity needed. I believe giving the reader a thread of their own will improve compressed input

---

[1] http://hem.passagen.se/hno/squid/squid-2.2.STABLE4.log_ip_on_direct.patch

[2] http://hem.passagen.se/hno/squid/

speeds, for a start. Also, creating disjunct sets of counters and feeding them to a work line might give some more throughput. Still, using for these parts a work crew might even further speed up processing.

On the other hand, looking at heterogenous computer pools in universities, or just multi processor nodes without shared memory, the MPI toolkit[1] 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 more flexible than threads.

Anyway, with almost 40000 lines per seconds, tendency increasing with 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 as is used to be.

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.

---

[1] http://www.mpi-forum.org/ and http://www.globus.org/mpi/

6 Selected Internals

This section will deal in a few choice internals.

6.1 Parsing an URL

The URL is a user entered input, and may thus contain grossly malformed input. Figure 1 shows the finite state automaton which extracts the method, host name and port from an URL. Start state is state 0. The final state 5 is not explicitly shown.

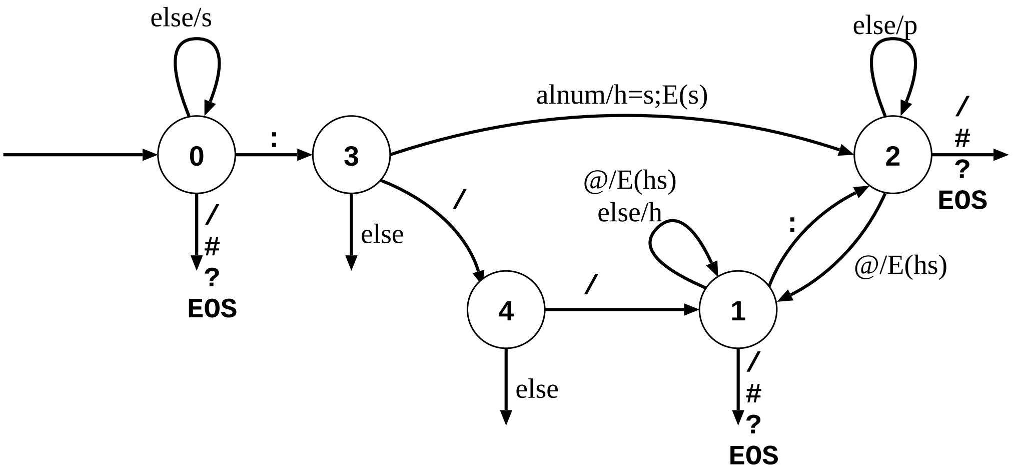

Figure 1: Finite state automaton for parsing URLs.

The connotation at the arcs may look a little weird. I apologize for not using standard notation. The Mealy automaton preforms 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 -
```

6.2 Reading log files

Figure 2 shows the class hierarchy of the log file reader. The design is neither good nor beautiful, but it is efficient.

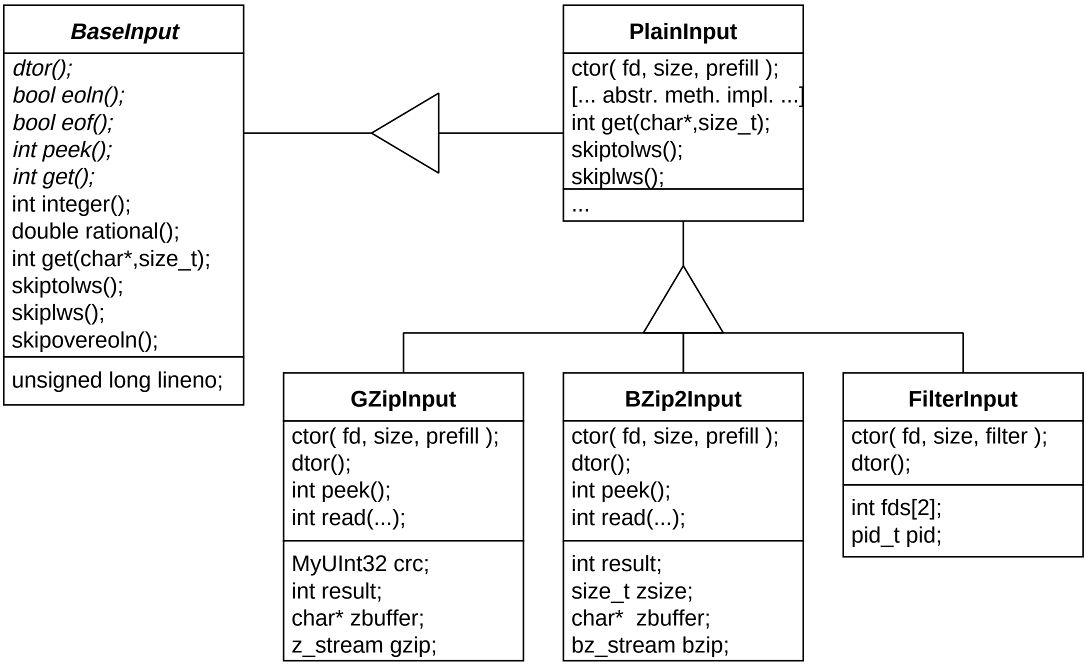

Figure 2: Class hierarchy for log file input.

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 2. 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 most certainly will 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.

`peek()` is access to the look-ahead character, and includes the states error as -1 and end-of-file (EOF) as -2. `get()` obtains the character with 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 needed 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. Also, it overwrites almost all base class methods with a more efficient refinement.

`GZipInput` is logically a sibling of `BaseInput`, and was planned as brother to `PlainInput`. Implementation though showed that is was easier to implement as 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 prcoessor (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. Putting the input classes into threads of their own should always be considered beneficial, though.

6.3 Trie matcher

The central data structure enabling the analyser to its perceived speed is the trie[1] data structure. 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, compare the trie in figure 3 which contains the words nag, nave, navy, nose, no, none, noon and now.

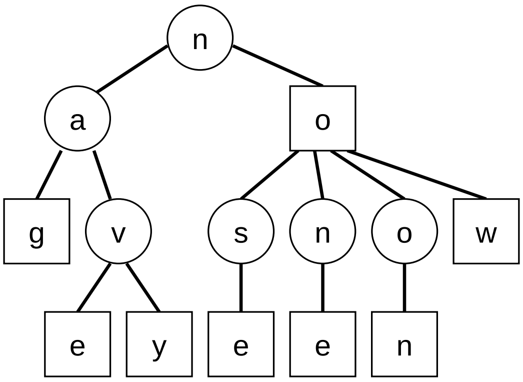

Figure 3: Trie of the words nag, nave, navy, nose, no, none, noon and now.

The trie nodes matching a word are shown as squares, non-word nodes are shown as circles. Please note that a word matching node is not limited to leaf nodes.

---

[1] originally from re-trie-val, but in order to distinguish from a tree, often pronounced as try-ee.

When comparing the match function of a trie with a hash table, both (usually) need a time proportional to the length of the word looked at. But as a trie by then has arrived at the stored value, a generic hashes usually need to do collision avoidance.

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 though will need to touch all letters of a mismatch and possibly do some collision avoidance in order to conclude a mismatch. Worse, if the hashes collision avoidance is coded unfortunately, each letter of the word will be need to be touched repeatedly for string comparisons.

The drawback of a trie is the insertion of new elements, which takes considerable more time than an insertion into a generic hash. Therefore, tries are used for static information which is read in once from the config 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. 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 seconds 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 loads 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 a little slower.
4. The not used implementation trie to be both, memory efficient, unlimited and fast. It uses a growable vector approach, and a character table indirection. Unfortunately, it does not work (yet), and was thus excluded from the project.

6.4 The string implementation

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. Figure 3 sketches the interrelation of the different classes having to do the handling of symbolic information.

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[1]. As the class is abstract, thus shown in italics, a sibling

---

[1] The idiom *interface* is taken from Java, but appropriately describes the classes function.

must supply the abstract method, if it does not want to be abstract itself. The `Hashable` interface is common, and thus needs some refinement with regards to strings.

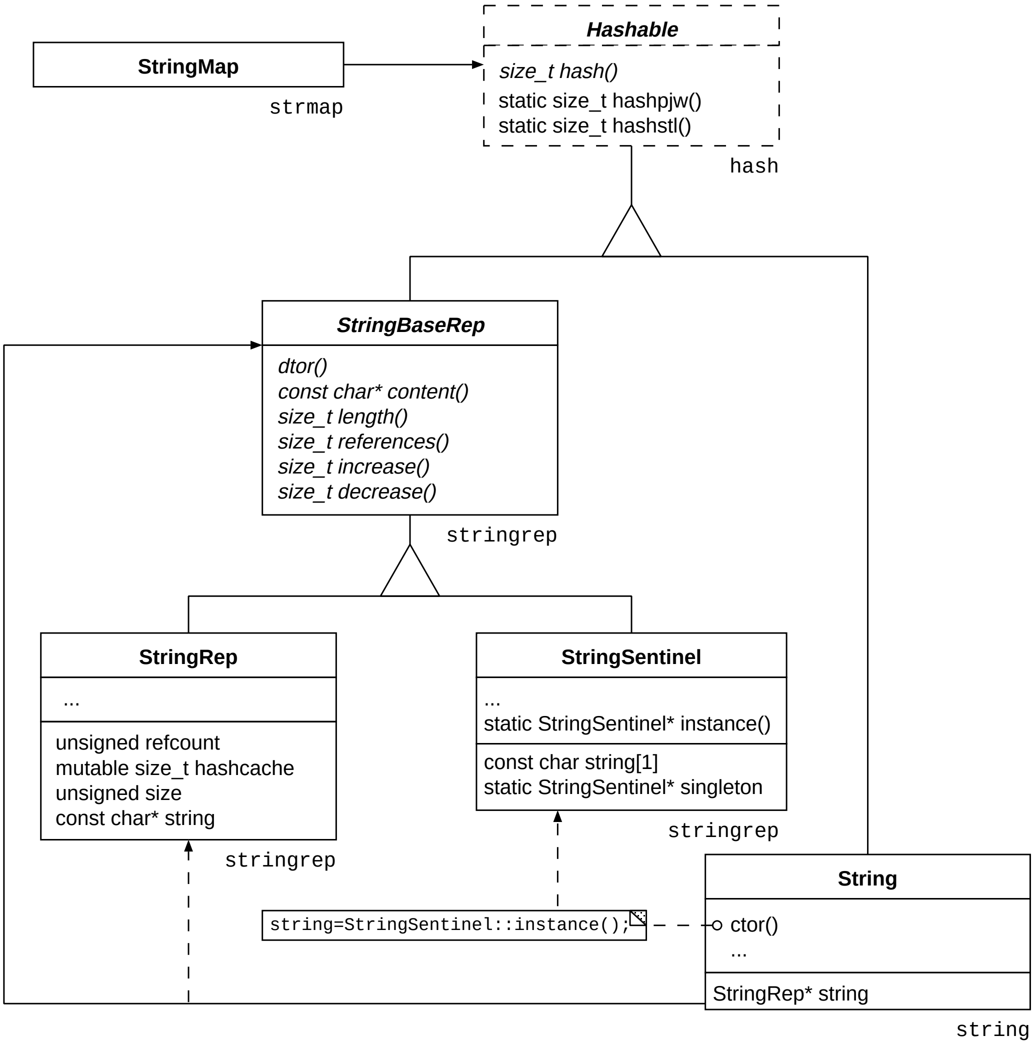

Figure 4: Chart of string class interrelations.

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, and 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 respective. 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 arithmetic 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 dare-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.

A Deviations from Deliverable D1

Compare these list of deviation with Deliverable D1 for the project[1]. The following items were changed, omitted, or modified as compared with the system analysis presented in the above document.

A.1 Deviations from section 1

A.1.1 Intervals

Interval I_1 deals with hourly peaks, and is implemented configurable, see the `peak_interval` parameter. Interval I_2 deals with all other values which are not peak related. I_2 is set by the configuration option `daily_interval`, but still needs to be implemented, and will be as soon as the analyser uses a cache for its internal counters.

A.1.2 Time output

The time format vs. bandwidth format is a configurable option, see `prefer_datarate`.

A.1.3 Squid 2 based log 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.

A.2 Deviations from section 2

A.2.1 Peak values

The so-called peaks are a kind of performance data. The peaks do not show the miss and none values for TCP, but show the *direct*, *parent* and *peer* values instead. I believe these are more feasible than the originally suggested approach.

A.2.2 Domains

The domain list is split into top level domains and 2nd level domains.

A.2.2.1 Top level domains

The top level domain list is limited by a list of correct domains which is configurable. Still, it contains over 240 correct top level domains. Two virtual top level domains were introduced. `<numeric>` is used for dotted quad destinations and also for single bigint destinations. The latter are a new feature of the BSD resolver library. Most `inet_XXXX()` functions convert them automagically into a correct internet address.

The virtual top level domain `<error>` is used for erroneous URLs, in which a host name could not be found or not be parsed. Also, it will be used for domains not in the top level domain list.

---

[1] http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable1.pdf

A.2.2.2 2nd level domains

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.

From practical judgement, e.g. seeing over 27000 distinct 2nd level domains per day and log file, it seems infeasible 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 admins option, since they are easily extracted, but due to the amount of data never be put into any kind of database.

A.2.3 MIME types

The MIME types, also known as media types and media sub types, are proliferating like the 2nd level domains. For that reason, the media types and media sub types of interest are configurable, and thus limited, to those mentioned in the configuration file. Any media type or sub type not in the lists is counted as `<unknown>`.

A.2.4 Request methods

Unknown (not configured) request methods are logged as `<unknown>` and not `<error>`.

A.2.5 Server side overview

The (first) coarse overview of the server side forwarded requests contains a hit count, too. Usually, one should not see any hits for server side requests. Since the configuration file can be malconfigured, this is a kind of fail-check. Apart from that, a distinction in HIT, MISS and NONE/ERR count as the introduction to D1 calls for, is not feasible.

A.3 Deviations from section 3

Section 3 of deliverable 1 deals with optional material. Please note that [RFC 2396] is now the correct literature to cite, not [RFC 1738].

A.3.1 pure HITs

The requirement looks as if met implicitly by 2.5, but it is not. Since the analyser can be misconfigured to count strange lines as hit, a pure hit count would be nice. Still, I do not deem the effort worthwhile, since correctly implemented analyser does display those hits which were called "pure".