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

Seafood - a log file analyser (Deliverable 2)

Jens-S. Vöckler 10. August 1999

Table of contents

1 Compiling Seafood . . . 2
 1.1 Installing compression libraries . . . 2
 1.2 GNU make, flex and bison . . . 3
 1.3 NDBM . . . 4
 1.4 The actual compilation . . . 4
 1.5 Installation . . . 4
2 Configuration options . . . 5
 2.1 List item aliasing . . . 6
 2.2 The method_list . . . 6
 2.3 The hierarchy_list . . . 6
 2.4 The status_list . . . 6
 2.5 The scheme_list . . . 7
 2.6 The mediatype_list . . . 7
 2.7 The media_subtype lists . . . 7
 2.8 The domain_list . . . 7
3 Running Seafood . . . 8
 3.1 Stderr . . . 8
 3.2 Output: writing the results . . . 9
4 Some questions and answers . . . 12
5 Missing things (to do list) . . . 13
 5.1 What is a HIT, what is a MISS, what is the rest? . . . 13
 5.2 Meta Traffic . . . 13
 5.3 Gaps in the log file . . . 14
 5.4 Configurability . . . 14
 5.5 Lookup suppression . . . 15
 5.6 Reading stdin . . . 15
 5.7 Table length . . . 15
 5.8 URL parsing . . . 15
 5.9 Multiprocessing . . . 16
6 Selected Internals . . . 17
 6.1 Parsing an URL . . . 17
 6.2 Reading log files . . . 18
 6.3 Trie matcher . . . 19
 6.4 The string implementation . . . 20
A Deviations from Deliverable D1 . . . 23
 A.1 Deviations from section 1 . . . 23
 A.2 Deviations from section 2 . . . 23
 A.3 Deviations from section 3 . . . 24

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 a very early version of the parser. It is known to parse Squid 2.2.STABLE4 access.log files. Older versions of Squid, even Squid-2.0.x or Squid-2.1.x, will need some changes in the configuration file. But I am getting ahead of myself here. Ok, let's start at the beginning.

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, or a native C++ compiler on a Unix platform which understands about `mutable`, `bool` and for-scoped loop variables. 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

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

Table 1: Systems and compilers known to work.

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.

Seafood was successfully translated on the systems mentioned above, though with Irix, only the native compiler will do the trick. The g++ 2.8.1 series should be able to compile the project on many systems not mentioned above.

1.1 Installing compression libraries

The seafood log file processor can read compressed files, though this is **not** recommended on multi processor machines. 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, e.g. comment it out in section 1 of the `Makefile`.

1.1.1 libz

Refer to the zlib homepage[3] 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 cor-

1. http://www.cache.dfn.de/DFN-Cache/Development/Seafood/
2. http://calamaris.cord.de/
3. http://www.cdrom.com/pub/infozip/zlib/

rect 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.

1.1.2 libbz2

Refer to the bzip2 homepage[1] 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.

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 out for modifications 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.

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

1. http://www.bzip2.org/

system library is sufficiently fast:

```
$ ./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 NDBM

For caching DNS and internet routing registry (IRR)[1] replies, a NDBM based cache is used. Also, the output phase of subsequent runs is thus sped up. Linux uses the BSD DB emulation employing `libdb`, Solaris and Irix have NDBM built into their respective libc. Other systems might need to add the flag `-lndbm` to the LOADLIBES macro in the `Makefile`.

1.4 The actual compilation

Just say make and watch the compiler fiddle away. With Linux, Solaris and Irix, some optimizations are supplied. Some parts of the code can cause a g++ to break, thus for the counters.cc module all optimization is turned off 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 g++ or egcs, the compiler option `-mcpu=pentiumpro` will really make things still a little faster.

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

1.5 Installation

If you are planning on parsing Squid-2.2.STABLE4 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 RA whois mirror[2].

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. also known as the `whois` service.
2. http://www.irrd.net/

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
peak_interval	<int>	3600
warn_crash_interval	<int>	1800
log_fqdn	<bool>	true
dns_cache_file	<string>	"/var/tmp/dns"
dns_positive_ttl	<int>	18144000
dns_negative_ttl	<int>	604800
irr_cache_file	<string>	"/var/tmp/irr"
irr_server	<string> | none	"whois.ra.net"
irr_positive_ttl	<int>	18144000
irr_negative_ttl	<int>	604800
method_list	<id list>	section 2.2
warn_unknown_method	<bool>	false
hierarchy_list	<id list>	section 2.3
warn_unknown_hierarchy	<bool>	false
status_list	<id list>	section 2.4
warn_unknown_status	<bool>	false
scheme_list	<id list>	section 2.5
warn_unknown_scheme	<bool>	false
mediatype_list	<string list>	section 2.6
media_subtype	<string> <string list>	section 2.7
domain_list	<string list>	section 2.8
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 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.

2.2 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 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.

2.4 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.

2.5 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.

2.6 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).

2.7 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>`.

2.8 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 13).

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, see section 5.6 (page 15) on the issue of `stdin`.

A typical run 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.

```
# trying to read "./seafood.conf"
# ...10 ...20 ...24 ...34 ...42 ...48 ...53 ...58 ...66 ...72 ...77
# ...109(18) ...114 ...146(38) ...151 ...206(19) ...211 ...230(12)
# ...271(12) ...295(5) ...305(7) ...321(5) ...329(3) ...335(2) ...348(6)
# ...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.

```
# 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.

```
45643: neither TCP nor UDP
+ 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 13). 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 was processed, the output will be generated. Each section for the output has a respective log associated:

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

```
# stats...
# DONE!
```

DONE signals the finish of the output. 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.
ans...	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.
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 lookups, 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*.

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.

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>`.

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>`.

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, currently you need to change the `main()` function in `seafood.cc`, sorry for this.

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 are currently limited to the top twenty. If you want to limit the output to a different number, or use unlimited output (0), currently you need to change the `main()` function in `seafood.cc`, sorry for this.

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 are currently limited to the top twenty. If you want to limit the output to a different number, or use unlimited output (0), again you need to change the `main()` function in `seafood.cc`.

The AS information will be obtained by opening a persistent connection to the whois server and piping all requests, one at a time, to the server. The part is not particularly stable yet (e.g. sudden connection drops), and you might want to disable this feature in your seafood configuration file with:

```
irr_server none;
```

Waiting for the RA whois server is really slow, up to 30 seconds per request. For instance, processing a log file took 12 seconds, and waiting for the RA 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].

The whois mirror server can also be used for Squid, see `as_whois_server` in your `squid.conf`, that is, if you are using the `dst_as` and `src_as` ACLs.

Important: Please note that the RIPE whois server does **not** understand the extended RA syntax. You **must** use a RA or Merit compatible whois server.

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.

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

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.

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 is currently limited to the top twenty. If you want to limit the output to a different number, or use unlimited output (0), you need to change the `main()` function in `seafood.cc`, sorry for this.

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 is currently limited to the top twenty. If you want to limit the output to a different number, or use unlimited output (0), once again you need to change the `main()` function in `seafood.cc`.

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. A third could deal with time.

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).

3.2.14 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.

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. 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 13) 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 lookup.

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.

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

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 23).

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, provocatively called worthless, should be displayed separately, in order to give an administrator some kind of figure to see

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

what percentage traffic was 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.

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.

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 Lookup suppression

In some instances, a form of output format is needed which may or may not need DNS lookups, 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 lookups 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. If then the Squid had the (not recommended) `log_fqdn` option activated, forward lookups 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 lookup 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, a DNS forward lookup can be avoided.

5.6 Reading stdin

For multi processor machines, it is recommended to use a separate decompression and pipe the output into the analyser. Unfortunately, it does not (yet) support the dash - file name, meaning `stdin`. On some systems, you can fool it either by using the `/dev/fd/0` file name, or using a named pipe.

5.7 Table length

The tables for client side accesses, top level domains and second level domains are currently hard coded into `seafood.cc`, and should be run time configurable. Please mind that this is an early release.

5.8 URL parsing

The finite state automaton described in section 6.1 (page 17) has one obvious weakness. It is unable to correctly determine the scheme, host name or port, if any of those contains URL

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

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.

5.9 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 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.

Anyway, with over 20000 lines per seconds, tendency increasing with more powerful platforms, parsing a days worth of log files is a matter of minutes, and thus multi threading 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.

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.

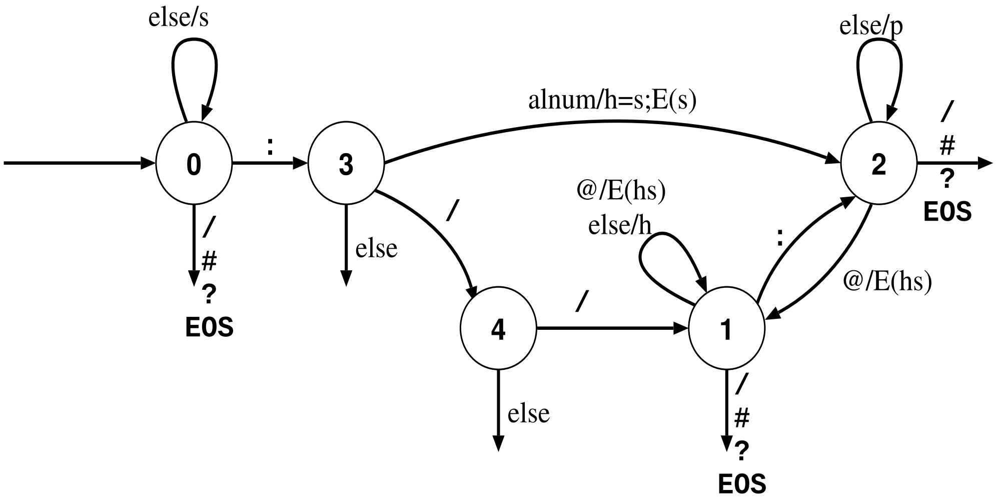

Figure 1: Finite state automaton for parsing URLs.

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.

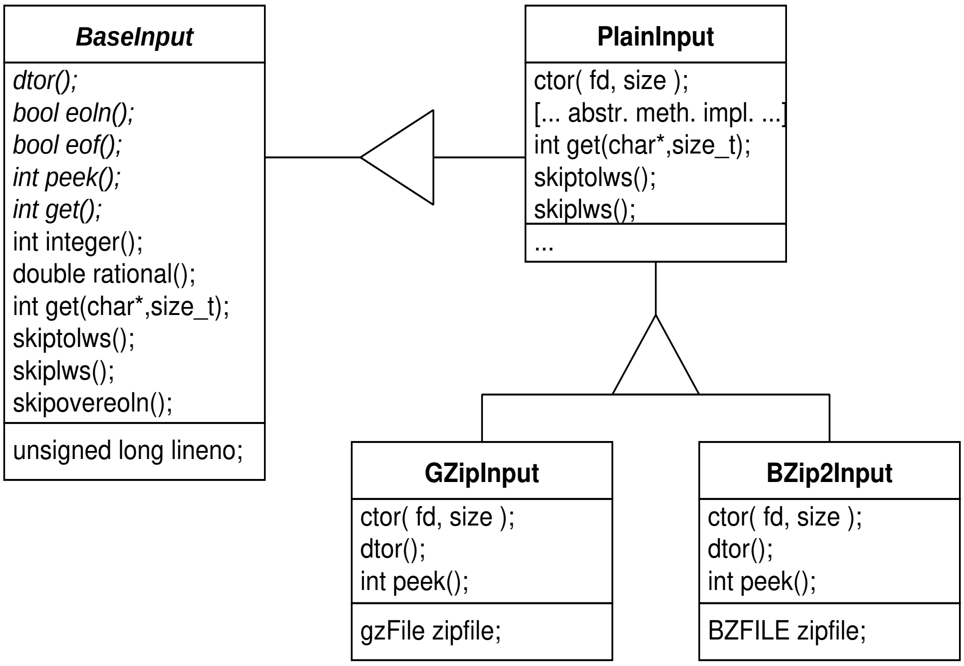

Figure 2: Class hierarchy for log file input.

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 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. On a single CPU system with a decent scheduler the internal unzip mechanism might lead to less contending context switches, and might thus improve performance. Putting the input classes into threads of their own should always be considered beneficial.

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.

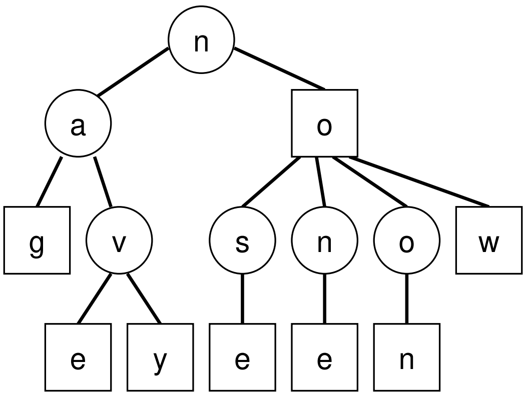

Figure 3: Trie of the words nag, nave, navy, nose, no, none, noon and now.

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.

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

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

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 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.

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

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

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.

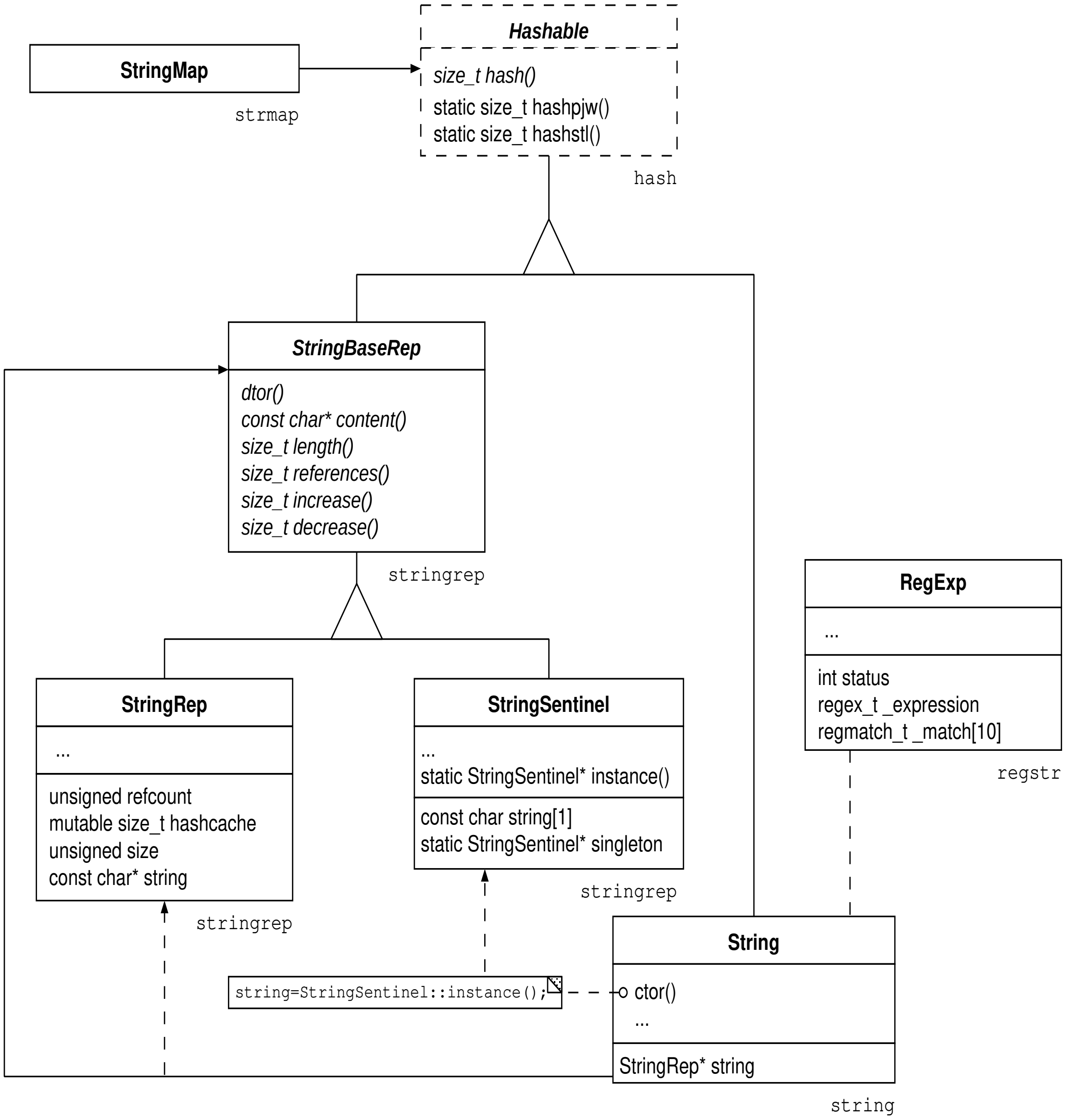

Figure 4: Chart of string class interrelations.

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.

The association between `String` and `RegExp` is very loose. The latter uses the string class to return certain results like matched sub expressions. The String class uses a regular expression wrapper to do sed like substitutions. The are no longer necessary in the project, since regular expressions are extremely slow when compared with trie matches.

A Deviations from Deliverable D1

Compare with Deliverable D1 for the project[1]. The following items were changed/omitted/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 needs yet to be implemented, and will be as soon as the analyser uses a cache for its counters.

A.1.2 Time output

So far, the analyser prints a bandwidth output. That is not a feasible output for the database. D1 explicitly states that a time should be used, as the calculations of bandwidth are done using the database interface.

The time format vs. bandwidth format and should be a configurable option, so the analyser can also be used by people who are not interested in long term storage of their results.

A.1.3 Squid 2 based log files

So far, the analyser is known to parse Squid-2.2s4 log files. Effort and time must be spent to parse elder versions of Squid-2 log files.

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.

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 puts them 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 Client side overview

No changes.

A.2.6 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.2.7 Detailed client side

The UDP traffic meets the requirements. The TCP traffic meets the requirements, too, but is a little awkward to read (as a text file).

A.3 Deviations from section 3

Section 3 of deliverable 1 deals with optional material.

A.3.1 Destination autonomous system

It is implemented, and requires access to a RA or Merit compatible whois server.

A.3.2 Distributions in size and time

Not (yet) implemented, but can be easily done.

A.3.3 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 in D1.

A.3.4 Protocols

The output of the URL schemes is configurable, and part of the analyser. [RFC 2396] is now the correct literature to cite, not [RFC 1738].