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

Seafood - a log file analyzer (Deliverable 4)

Jens-S. Vöckler 12. January 2000

---

1 The prototypical web interface

The last phase of the extended cache statistics project deals with the prototypical implementation of a web interface to the database designed in phase 3. Besides querying for the plain results of a single table stored within the database, the data of greater interest can be found by combining and grouping more than one table. Figure 1 is an example for such a powerful combination. Details on figure 1 can be found in section 1.5.

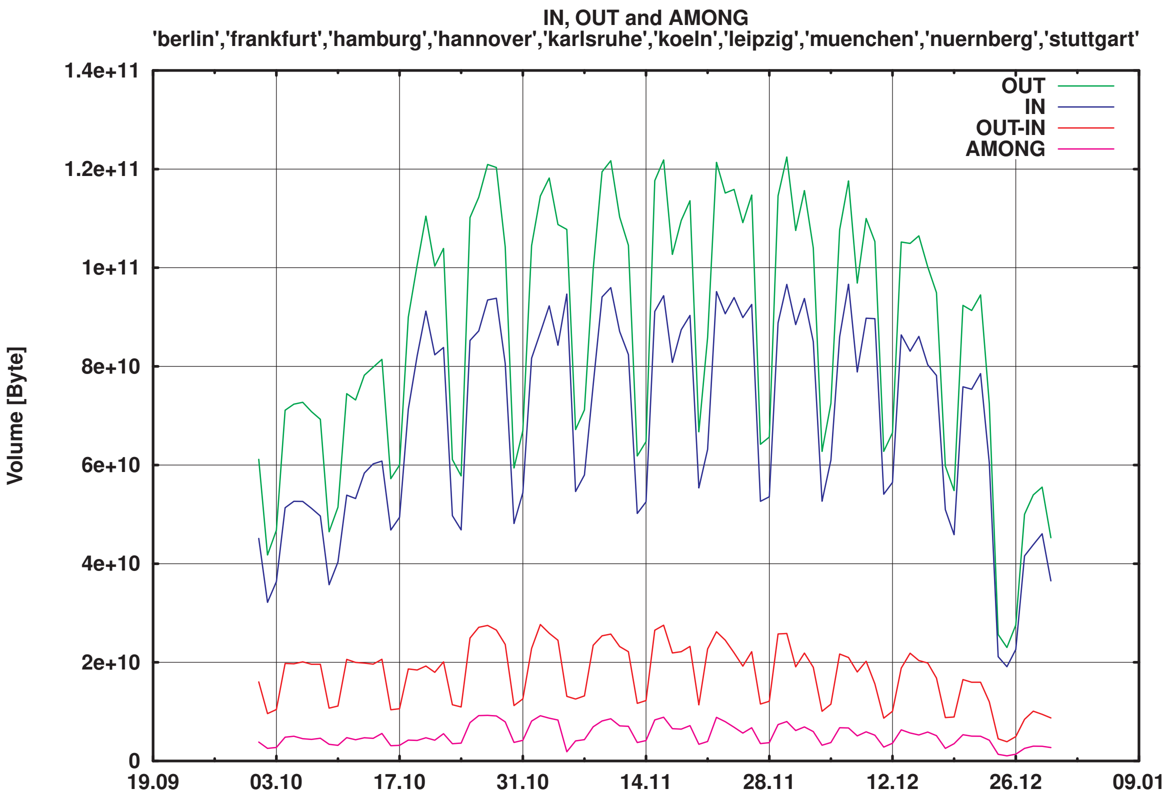


Figure 1: Example for a combined extract from 10 cache hosts.

The current section starts out with an introduction into the web based interface. It continues by giving an overview of the implemented options on an abstract basis. Later sections will deal with the concrete implementation.

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.

1.1 The selection menu

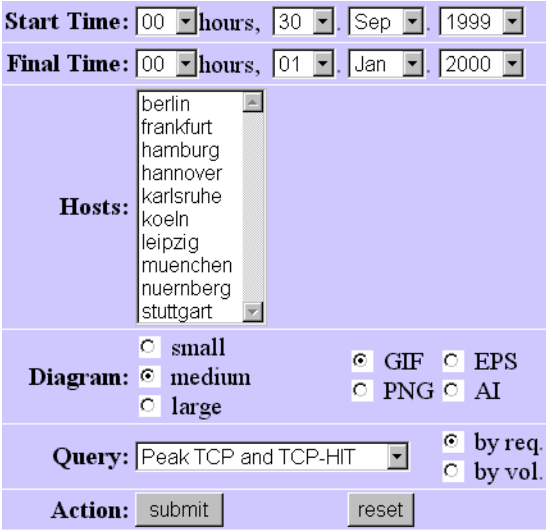


Figure 2: Example for the web based interface.

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

The third part lets a user select a cache host. The cache host list is also generated by a query to the database. Please note that at least one cache host must be selected.

The fourth section deals with the canvas size and the media type of the output. The media types can easily be extended to any gnuplot supported graphics format.

The fifth secion determines the kind of query. The prototypical implementation contains nine different options, which will be covered in the following sections. The kinds of query can easily be extended to options of other interests by extending the perl scripts. Also, a selection by requests and by volume is part of the selection.

The final column either submits the choices to the next stage, or resets the form to its default.

1.2 Peak TCP and TCP HIT

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

1.3 Non-GET method sums

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

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

1.4 Special TCP clients

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

1.5 IN, OUT and AMONG

The IN, OUT and AMONG option allow for a more powerful combination than the ones previously shown. It allows to evaluate the efficiency of a cash mesh, the savings as OUT-IN, and the overhead, the traffic AMONG the caches. Refer to figure 1 for an example output. Unfortunately, the traffic bypassing the cash mesh, labelled with a question mark in figure 3, cannot be computed, as it does not appear in the cache log files.

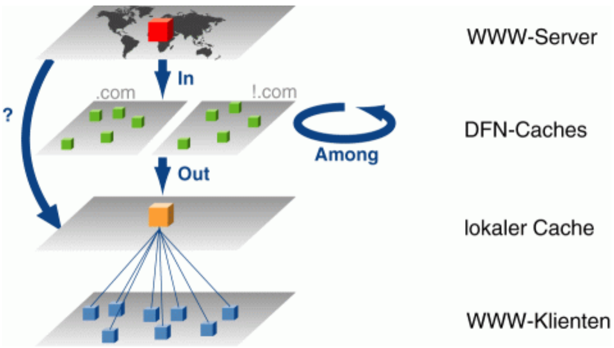


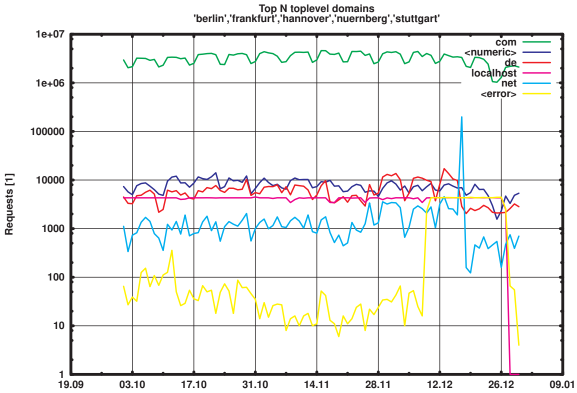
Figure 3: Efficiency of a cache mesh as a source.

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

The savings OUT-IN are graphed separately in order to make the savings comparable to the overhead. Figure 1 shows the savings and overhead for the DFN caches starting from October 1999 till the end of the year.

1.6 Frequent vs. top N top-level domains

There are two options dealing with the top-level domain names. The *frequent* option uses a fixed set of frequently encountered top-level domain names. The *top-N* option on the other hand dynamically determines the six most common top-level domains for the specified interval from the top-level domain table. Please note that viewing by requests and viewing by volume might yield a different set of top-N domains for an identical interval.



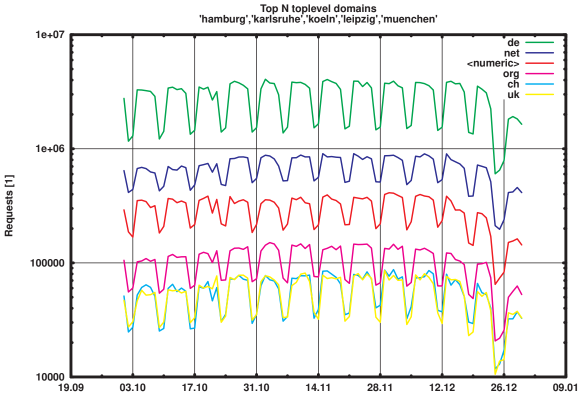


Figure 4: Example for top-N domains of *.com and !.com hosts by requests.

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

1.7 Frequent MIME types

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

1.8 Top N autonomous system (AS) destinations

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

1.9 Top N clients

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

1.10 Hierarchy performance

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

2 Update of the Database Design

This section will show the update of the database design, because some flaws in the first database design[1] were discovered during the implementation of the web based interface.

The improvements of the database design are shown in figure 5 as updated model of all tables which depend on the daily[2] interval I_1. Due to the less abstract notation, Postgres data types were used instead of SQL types. As mentioned during the database design phase, the volume results should be stored using at least 64-bit integers. The design was optimized in favor of less tables, as it is easier to select and group results from a single table than to aggregate from different tables. A single char was chosen as boolean type for predicates.

The new design still shows a circular arrangement of tables around the central table `st_stamp1`. The primary key fragment `I1ID` is an abstract number. Table `st_stamp1` uses an alternate key consisting of the combination of the cache host name and the *beautified* start time stamp `FIRST` of interval I_1, the real key for the central table. The real start time stamp is contained in the `BEGIN` column. The `I1ID` is the glue between the tables shown in figure 5.

Due to the lacking generic support in some minor databases, sequences and triggers are visible in the design. A redesign should focus on a powerful generic abstract database and leave these details to be programmed, if necessary, in the database actually used. In this regard, the database design is not as optimal as it could be nor as abstract as it should be.

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

The top of figure 5 shows the tables dealing with the server side traffic of the squid cache. The hierarchy code is part of the primary key. For a sibling and a parent, the host contacted is also part of the key. The formerly separate tables for siblings and parents are now collapsed into one `sf_peer` table. The `IS_PARENT` predicate is redundant again, because the hierarchy code sufficiently distinguishes between parent and sibling.

The top-level and 2nd-level domain tables could theoretically be combined. But as the administrator is expected only to store the top N 2nd-level domains, the top-level domain information is not completely contained in the 2nd-level domain table.

1. Deliverable 3 (http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable3.pdf)
2. Deliverable 1 (http://www.cache.dfn.de/DFN-Cache/Development/Seafood/deliverable1.pdf)

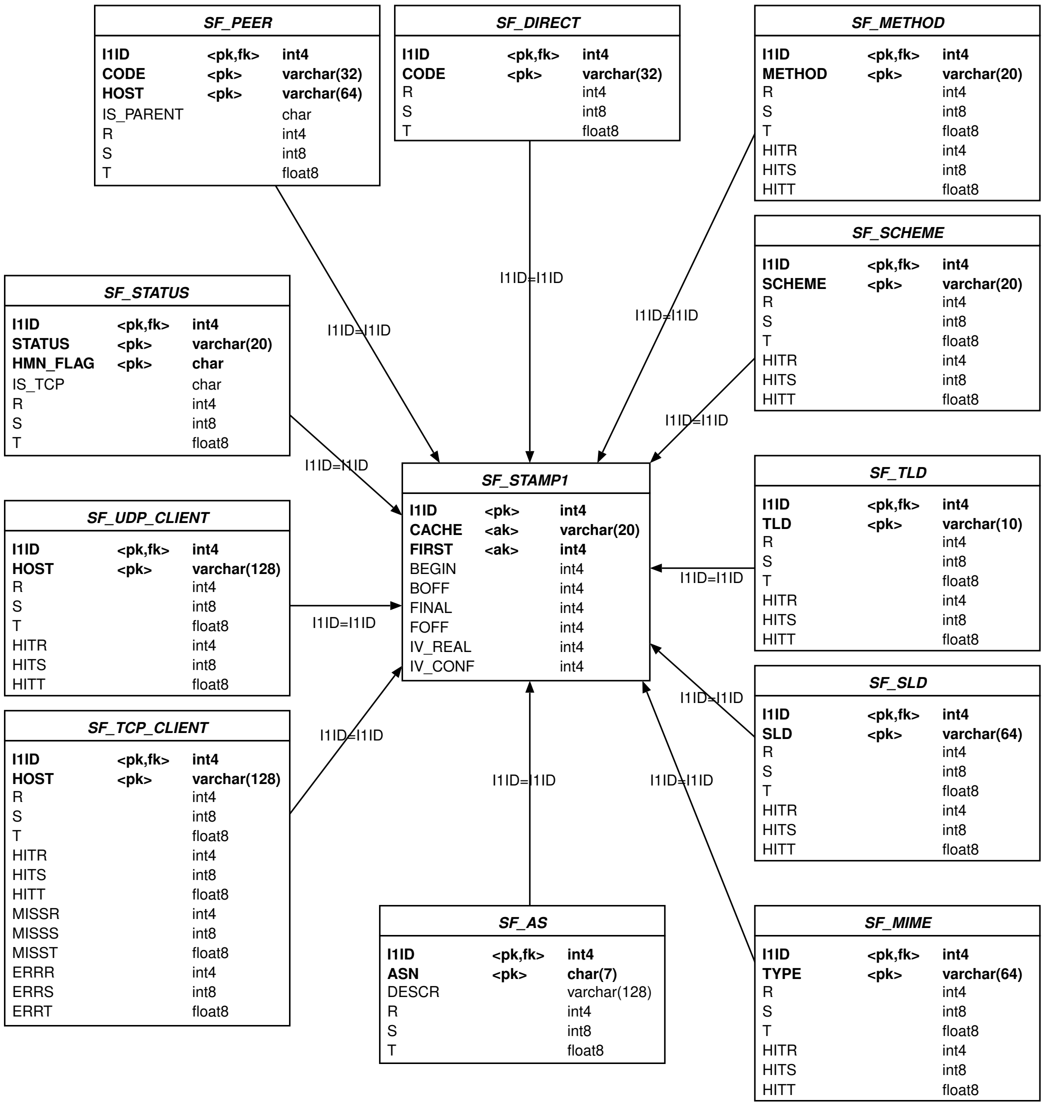


Figure 5: Physical model of tables depending on the daily interval.

There is no HIT information to the `sf_as` table, because the destination AS information is obtained from the direct fetches.

Figure 6 shows the `sf_stamp2` table, which deals with the hourly interval I_2. Only the performance statistics table `sf_peak` depends on the second stamp table.

The `sf_meta` table records the highest number of the interval IDs in a programmatic manner. A redesign should assume the availability of sequences, and leave this detail to the concrete implementation for a database.

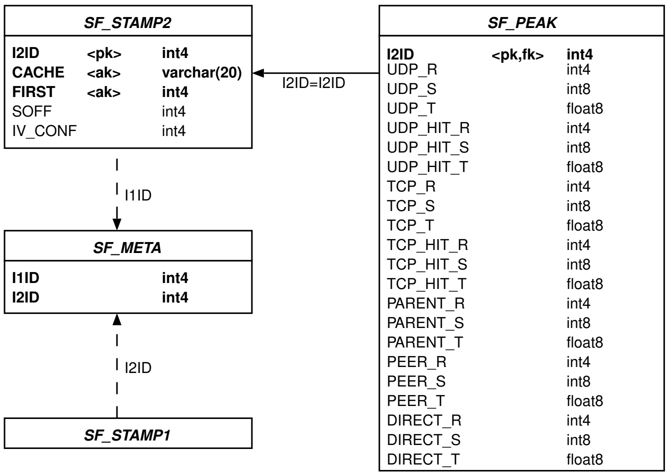


Figure 6: Remaining tables.

3 Implementation of the prototypical interface

The previous sections showed the interface on an abstract basis and the current database desing. This section will pull it all together, and describe how the parts of the prototypical web interface interact.

A web based gui was chosen due to its implementation independent access to the database underneath. Figure 7 shows the data flow diagram.

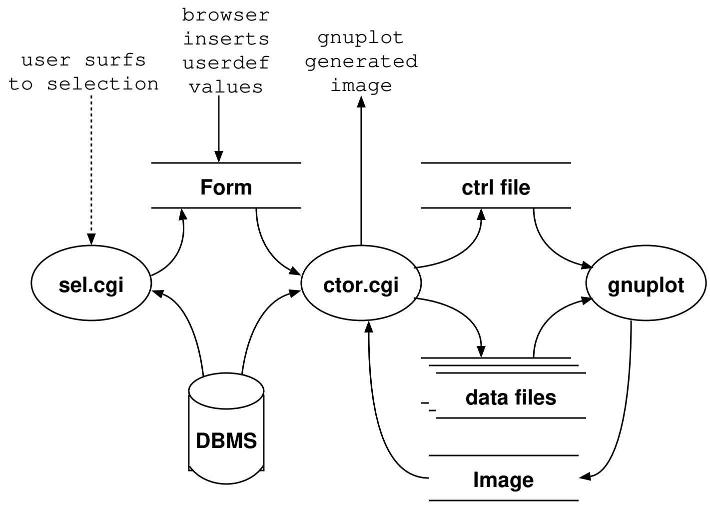


Figure 7: CGI data flow of web GUI.

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

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

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

3.1 gnuplot

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

3.2 Scripts.pm

The scripts Perl module contains helpful functions and data pertaining to handling CGI data. At start-up, the module changes the standard IO file handles to unbuffered mode.

The function `now()` with an optional argument converts the optional time stamp parameter into a HTTP header compatible time format. If an argument is missing, the current time will be taken.

The function `parseQueryString()` creates a Perl hash from form submitted data. Sanity checks are left to the calling instance, but necessary.

3.3 Wrapper.pm

The wrapper Perl module only exports the `database_login()` function. You will need to make changes here in order to suit your database environment; e.g. the database host and database user login and password most certainly need to be changed. Examples for Oracle and PostGres are shown as private functions within the module.

3.4 sel.cgi

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

3.5 ctor.cgi

The constructor script is a rather monolithic creator of images. You may need to adjust the path to your Perl and you should check the complete path to the modified version of gnuplot. Also, you might want to change the temporary directory mentioned earlier.

The scripts starts by sanitizing the user submitted values. If the values are o.k, more data structures are set up for later perusal. In the next step, the script connects to the database, creates the gnuplot control file, and calls a work function. Work functions are explained below. After the function finishes, the database connection can be closed, and gnuplot will be called. If gnuplot returns successfully, the image will be copied onto stdout. Please note that the database calls are bracketed within an `eval` block in order to catch errors.

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

4 Implementation of the SQL queries

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

Some Perl variables are visible in the queries. A better implementation would be using database stored procedures, and calls to them with variable arguments. Still, the current implementation is also flexible enough for a prototype. Refer to table 1 for commonly used variables. Further variables are described in the sections.

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

Table 1: Common Perl variables in `select` calls.

4.1 Selection script queries

The selection script queries the database in order to generate the appropriate form to present to the user:

```
select min(first),max(first) from sf_stamp2
select distinct cache from sf_stamp1 order by cache
```

The first query select the interval range stored within the database. It is assumed that the interval I_2 uses the same range as interval I_1. The second query selects all names of the cache hosts.

4.2 Peak TCP and TCP HIT

The selection of the peak TCP rate and HIT rate is straight-forward from the `sf_peak` table. For each selected cache host, the following query will be used:

```
select st.first, p.tcp_s, p.tcp_hit_s
from sf_stamp2 st, sf_peak p
where st.first between $start and $final
and st.i2id=p.i2id and st.cache=\'$cache[$i]\'
order by st.first
```

The output columns are stored in one file each, later subjected to gnuplot. Due to the finer granularity of the peaks table, the generated graphs are useful for small interval up to a few weeks.

4.3 Non-GET methods sums

The non-GET methods request is in its structure similar to the previous one. The group by clause condenses all methods to one value which are not GET nor ICP_QUERY:

```
select st.first, sum(m.s), sum(m.hits)
from sf_stamp1 st, sf_method m
where st.first between $start and $final
and st.i1id=m.i1id and st.cache=\'$cache[$i]\'
and not ( m.method=\'GET\' or m.method=\'ICP_QUERY\' )
group by st.first order by st.first
```

4.4 Special TCP clients

The list of special TCP clients tries to show the output of the caches with subtraction of any inter-cache communication. The query is similar to the OUT value in the next section. All cache hosts are excluded by domain name as well as by address. For your own cache mesh, you need to replace the DFN cache hosts with your own host name and address set.

```
select st.first, sum(c.s)
from sf_stamp1 st, sf_tcp_client c
where st.first between $start and $final
and st.i1id=c.i1id and st.cache=\'$cache[$i]\'
and (c.host like \'%.win-ip.dfn.de\' or
     c.host like \'193.174.75.%\')
group by st.first order by st.first
```

4.5 IN, OUT and AMONG

The current query is the first working on a set of selected cache hosts simultaneously. The output are just four graphs, showing

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

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

```
select st.first, sum(tc.s)
from sf_stamp1 st, sf_tcp_client tc
where st.i1id=tc.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and not ( tc.host like \'%.win-ip.dfn.de\' or
          tc.host like \'193.174.75.%\' )
group by st.first
```

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

```
select st.first, sum(hd.s)
from sf_stamp1 st, sf_hier_direct hd
where st.i1id=hd.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
group by st.first
```

The savings are calculated each time stamp at a time as OUT minus IN for the current stamp.

The overhead AMONG is obtained from the sibling and parent traffic in the `sf_peer` table.

```
select st.first, sum(hp.s)
from sf_stamp1 st, sf_hier_peer hp
where st.i1id=hp.i1id
and st.cache in ( $cacheset )
and st.first between $start and $final
group by st.first
```

4.6 Frequent vs. top N top-level domains

These two options use the same basic query to obtain the data values. For a given set of top-level domains, with the current one being `$tld`, each domain will be queried:

```
select st.first, sum(d.s)
from sf_stamp1 st, sf_tld d
where st.i1id=d.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and d.tld=\'$tld\'
group by st.first order by st.first
```

The frequent top-level domains is a fixed set containing "com", "net", "org", "de" and the pseudo-domain "<numeric>".

```
select d.tld, sum(d.s)
from sf_stamp1 st, sf_tld d
where st.i1id=d.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
group by d.tld
```

The top N selection determines the set of domains dynamically by using the previous query. For each of the top 6 of the result set, the first query determines the output.

4.7 Frequent MIME types

For each `$type` of the fixed set of content types 'image/gif', 'image/jpeg', 'application/octet-stream', 'application/pdf', 'text/html', 'text/plain', 'audio/*', 'video/*', a graph from the chosen host set and interval will be drawn.

```
select st.first, sum(mt.s)
from sf_stamp1 st, sf_mime mt
where st.i1id=mt.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and mt.type=\'$type\'
group by st.first order by st.first
```

4.8 Top N AS destinations

The top N selection is two-fold again. The sum of all AS destinations with the exception of the pseudo destination *more* is determined in a first step:

```
select a.asn, sum(a.s)
from sf_stamp1 st, sf_as a
where st.i1id=a.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and not a.asn=\'more\'
```

```
group by a.asn
```

The top 6 destinations `$asns[$n]` are used to determine the course of the graphs for each:

```
select st.first, sum(a.s)
from sf_stamp1 st, sf_as a
where st.i1id=a.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and a.asn=\'$asns[$n]\'
group by st.first order by st.first
```

Please read section 5 for reasons why the shown volume selection is critical.

4.9 Top N clients

The four top N clients are using a similar set of queries. This section shows the HIT selection by volume for each given `$protocol`. In a first step, the top 6 hosts are determined from the appropriate client table, with the exception of the cache hosts themselves and the pseudo host *more*:

```
select c.host, sum(c.hits)
from sf_stamp1 st, sf_$protocol_client c
where st.i1id=c.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and not c.host like \'more[%]\'
and not c.host like \'%.win-ip.dfn.de\'
group by c.host
```

For each `$client[$n]`, the course of the graph is selected in a second step:

```
select st.first, sum(c.hits)
from sf_stamp1 st, sf_$protocol_client c
where st.i1id=c.i1id and st.cache in ( $cacheset )
and st.first between $start and $final
and c.host=\'$clients[$n]\'
group by st.first order by st.first
```

4.10 Hierarchy performance

The hierarchy performance is again a rather straight-forward request into the `sf_peak` table. The output is aggregated for the selected cache host set, and the course of the three graphs are plotted for the selected interval:

```
select st.first, sum(p.peer_s), sum(p.parent_s), sum(p.direct_s)
from sf_stamp2 st, sf_peak p
where st.first between $start and $final
and st.i2id=p.i2id and st.cache in ( $cacheset )
group by st.first order by st.first
```

5 Concluding notes

The database design could still be improved by eleminating even more tables and creating a more meta tables which save part of the configuration and known key words. Also, the database design should focus on an ANSI-SQL compatible abstract database, and leave the missing features to work-arounds in the implementation for a concrete database.

Any extension to the prototypical interface should note that the client tables, AS table and 2nd-level domain table only contain the top N by request items. Further items are summed up in the *more* pseudo entry. A selection as shown in section 4.8 is critical: If the user selects the volume output, some of the real high volume destinations may be hidden in the *more* pseudo entry, just because they happen to have a low request count.

A less fixed web interface would allow for more user interventions, e.g. when selecting the set of frequent items to show. It is thinkable that the web interface may work in several steps, like the step-by-step ordering from Amazon, starting from the common selections like interval, cache hosts and output format to special selections only valid for a chosen action.

5.1 Software

The seafood software bundle can be obtained from

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

The database scripts from phase 3 can be found in the `dbase` folder. The web interface scripts can be found in the `public_html` folder. Further information and pointers to documentation and binaries can also be obtained at the URL mentioned above.