MRTG – The Multi Router Traffic Grapher

Tobias Oetiker <oetiker@ee.ethz.ch>
Department of Electrical Engineering
Swiss Federal Institute of Technology, Zurich

August 31, 1998

Abstract

This paper describes the history and operation of the current version of MRTG as well as the Round Robin Database Tool. The Round Robin Database Tool is a program which allows to log and visualize numerical data in a efficient manner. The RRD Tool is a key component of the next major release of the Multi Router Traffic Grapher (MRTG). It is already fully implemented and working. Because of the massive performance gain possible with RRD Tool some sites have already started to use RRD Tool in production.

Motivation

In Summer 1994, the De Montfort University in Leicester, UK, had one 64 kBit Internet link for more than 1000 networked computers. As it was not possible to get a faster Internet link for another year, it was desirable to at least provide the users on campus with current and detailed information about the status of the link.

This situation prompted the development of the Multi Router Traffic Grapher. Every five minutes, it queried the "Octet Counters" of the university's Internet gateway router. From this data, the average transfer rate of the Internet link was derived for every five-minute interval and a web page was generated with four graphs showing the transfer rates for the last day, week, month, and year. The visual presentation on the Web allowed everyone with a web browser to monitor the status of the link. Figure 1 presents an MRTG-generated web page.

While the availability of these graphs did of course not increase the capacity of the link, the performance data provided by MRTG proved to be a key argument to convince management that a faster Internet link was indeed needed.

How MRTG-2 works

The original MRTG program was a Perl script which used external utilities to do SNMP queries and to create GIF images for display on the HTML pages. When MRTG was published on the Internet in spring 1995, it spread quite quickly and people started using it at their own sites.

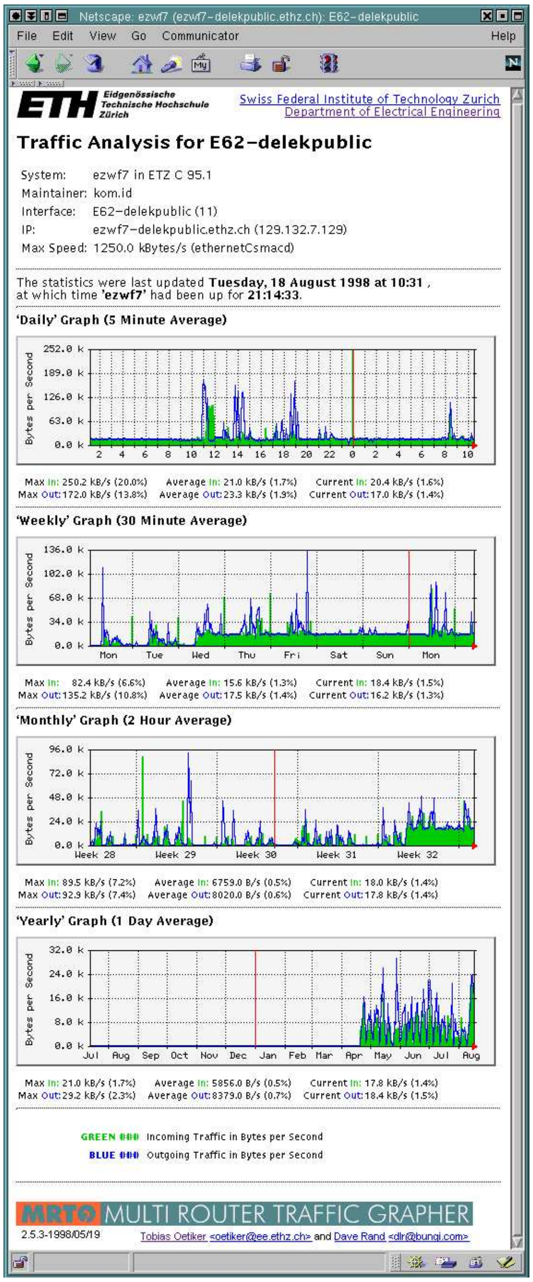

Figure 1: Screenshot of an MRTG-2 web page

Soon, however, user feedback highlighted two key problem areas: scalability and portability. While MRTG worked fine when monitoring 10 links, larger sites ran into performance problems. At De Montfort, the intention was to monitor the off-site Internet link and maybe one or two links between buildings and performance was not a limiting factor. External users pointed out though, that some sites had much larger monitoring needs and were running MRTG right at its limits.

MRTG logged its data to an ASCII file, rewriting it every five minutes, constantly consolidating it, so that the logfile would not grow over time. The logfile did only store slightly more data than was needed to draw the graphs on the web page. The graphs were converted to GIF format by piping a graph in PNM format to the `pnmtogif` tool from the PBM package. This setup limited MRTG to monitor about 20 router ports from a workstation.

A second obstacle for potential users was that MRTG required `snmpget` from the CMU SNMP package. This package proved to be rather difficult to compile on various platforms at that time.

In the mean time, I had left De Montfort University and was working at the Swiss Federal Institute of Technology. There I had no responsibility for the campus network and the Internet link was sufficiently fast. MRTG was not one of my top priority projects anymore. Because the CMU SNMP library did not compile on Solaris I had not even a working installation of MRTG.

This all changed when Dave Rand <`daver@bungi.com`> got interested in MRTG and contributed a small C program called *rateup*. *Rateup* solved MRTG's performance problem by implementing the two most CPU intensive tasks in C and thus moving them out of the MRTG Perl script. *Rateup* did the logfile rewriting and the graph generation.

Rateup initiated the development of MRTG-2.x. First, I modified *rateup* to use Thomas Boutell's GD library [5] which allowed to generate GIF files much faster than `pnmtogif`. Second the SNMP portability problem was solved by switching from the CMU *snmpget* to Simon Leinen's Perl SNMP module [4], which was written in pure Perl, making it virtually independent of the platform it was running on.

After almost a year of beta testing and the implementation of many user requested features, the result of these efforts was released as MRTG-2.0 in January 1997.

MRTG-2 was not only faster than the previous release, it was also more user friendly. A tool called *cfgmaker*, which is included in the MRTG distribution, is able to build a skeleton configuration file for a router by reading its interface table via SNMP. This allows a lot of people to successfully configure MRTG even when they do not know too much about SNMP or about how to find out which physical router interface is mapped to which SNMP variable.

MRTG-2 does neither require the PBM package nor an external SNMP gatherer anymore. This made the porting of the package very simple. Without using *autoconf* or any similar system, the package compiles on most Unix platforms. Even a port to Windows NT required only a few changes to the pathname handling and the calling of external programs. The most amazing thing with the NT port was that Simon Leinen's SNMP Perl module worked under NT without change.

MRTG-2 turned out to have the right mix of features to attract the interest of a substantial number of people.

Lossy Data Storage

A key feature of MRTG-2 is its method for maintaining logfiles. The basic assumption for designing the MRTG-2 logfile was that the interest in detailed information about the load of the network diminishes proportionally to the amount of time which has passed between the collection of the information and its analysis. This led to the implementation of a logfile which stores traffic data with a decreasing resolution into the the past. Data older than two years is dropped from the logfile. The resolution of the logfile matches the resolution of the graphs shown on the web page. This lossy logfile has the advantage that it does not grow over time and therefore allows unattended operation of the system for extended periods of time. Drawing the individual graphs is relatively fast because no data reduction step is required and thus disk I/O is minimized.

MRTG-2 logfiles are stored in plain ASCII. Each line starts with a time stamp followed by the corresponding traffic data. The file starts with the most current entry and ends about two years in the past. For processing, it is read as a whole, processed in memory and written back to disk. This happens for every single update as shown in Figure 2. Figure 3 shows how the values from the logfile are consolidated over time.

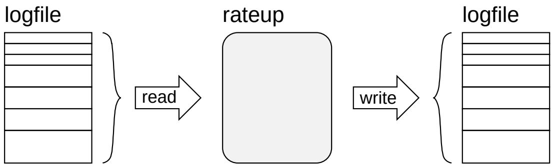

Figure 2: ASCII Logfile processing

Estimating the size of the user base

While it is easy to state that a substantial number of people are interested in a package, it is much more difficult to estimate many people really use it. Today, the MRTG home page gets about 700 hits and 200 downloads a day. These numbers are quite high for such a page, but it does not show how many sites are actually using MRTG.

Applications like MRTG, which produce output visible on the Web, offer a unique way to measure the size of their user base through the existence of the referrer header in HTTP requests. Every web page generated with MRTG contains a link to the MRTG home page. Whenever someone comes to the MRTG home page through this link, the web server of the MRTG home page logs the referrer header of the request. With this information it is possible to make a crude estimate about the number of sites using MRTG. Such an analysis was conducted in August 1998, using data from the last two years. It showed referrer headers from about 17500 different hosts under 11400 second level and 120 top level domains.

This excludes all installations of MRTG where the HTML output has been altered to not show the MRTG back-link as well as those where nobody has ever accessed the link. However, it still gives some sort of lower bound for the

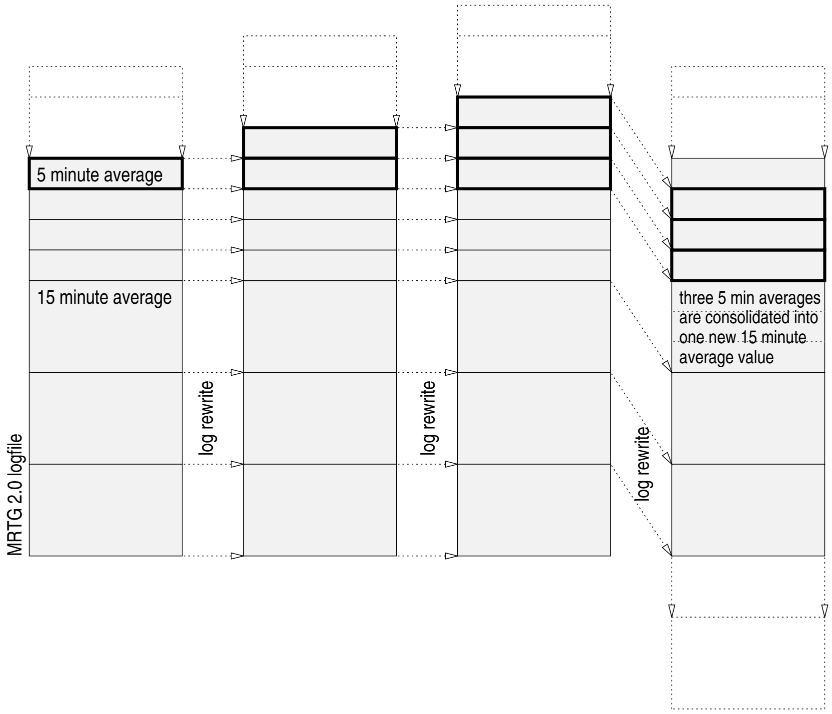

Figure 3: MRTG-2 Logfile processing

number of users.

End of life for MRTG-2

Because of the better performance, more and more large sites started to use MRTG. They soon hit the performance limit again, which was now at roughly 500 ports queried every 5 minutes. At the same time people started to use MRTG to monitor "non traffic" data sources, requiring more user control over the generated web pages and graphs.

While MRTG-2 initiated widespread use of the package, it was not fundamentally different from the original MRTG-1 Perl script. MRTG had just evolved to the point were it became useful for a larger community. For the maintainer of the package, on the other hand, this evolution had lead to a system which had outgrown its initial design. Every further enhancement added unproportionally to the complexity of the software. In November 1997, a complete redesign of MRTG was initiated. While some features would be totally new, old strengths were to be preserved.

User feedback and personal experience showed that the following features are the key elements to the success of MRTG-2:

Simple Setup: The configuration is done through simple ASCII text files. An additional tool helps creating an initial version of the configuration file, tailored to a certain router.

Easy Maintenance: Because the logfiles are automatically consolidated on every run and therefore do not grow in size, the system can work unattended for months without running out of disk space.

Friendliness: The HTML pages created by MRTG are easy to understand and give a good visual representation of the network load, providing a sound basis for decisions about upgrading network links.

Integrated Solution: MRTG performs all the tasks required for traffic monitoring. No external database or SNMP packages are required to make it work.

The main problem areas in MRTG-2 are the following:

Performance: MRTG-2 can not monitor more than about 600 router ports in a 5-minute interval, which is due to the way the logfiles are updated as explained above.

Flexibility: While MRTG-2 is quite configurable in general, this seems to make the users especially aware of the areas where configurability is limited, in particular when using the program to monitor time-series data other than network traffic.

The fact that people started using MRTG for tasks it was never designed for, going to great lengths tweaking it to get what they wanted, showed that MRTG offered a unique feature by integrating data collection, storage, consolidation and visualization in a single package. The goals for MRTG-3 were therefore set to be *flexibility* and *speed*.

Design and Implementation of MRTG-3

MRTG-3 moves away from being an application for monitoring network traffic only. The new MRTG will be a toolkit to build applications which monitor large numbers of diverse time-series data sources using a fast data logging facility. It will be able to create a wide variety of graphs, based on data gathered from one or several sources. A parallel SNMP gatherer will help to increase the efficiency of the SNMP data gathering process. The time-critical parts of MRTG-3 are implemented in C, while the glue of the package remains Perl. This allows the users to tailor the package to their needs without recompiling it.

The Round Robin Database Tool

Development of MRTG-3 started with the implementation of a completely new mechanism for data storage. It is called Round Robin Database (RRD), which gives a clue on how data is stored. The data handling as well as the generation of graphs is implemented in a C program called `rrdtool`. It can either be called from the command line or through Perl bindings.

The Round Robin Database is so much faster and more configurable than MRTG-2 that a number of people have started to use it in their own custom monitoring applications without waiting for the remaining parts of MRTG-3 to be written. After some discussion on the MRTG developers mailing list it was decided to spin off the `rrdtool` into a new package separate from MRTG-3, as it is a complete and useful product all on its own.

Existing software is more useful than planned features. Therefore the remaining part of this section will focus on the Round Robin Database Tool and touch on the other features of the MRTG-3 package only at the very end.

Database Design

Designing a new file format offered the possibility to include a host of features to make the new logfile not only faster but also much more flexible than the old text-based logfile from MRTG-2.

- The RRD format uses *doubles* for data storage. This gets rid of the integer overflow problems seen when monitoring really fast routers with MRTG-2 and it allows to log small numbers like the load of a machine without scaling.
- The RRD can also store *unknown* data values. Which allows to distinguish between situations where the data input is zero and those when no new valid data can be obtained.
- Parameters like the number of log entries, the resolution of the log and the number of data sources to log in parallel are configurable.
- The data values in the RRD are stored in native binary format. This makes access to the data more efficient, because no conversions are necessary anymore. A cookie in the header of the RRD is used to test if the RRD is compatible with the architecture it is being read from.

Data storage in an RRD is a multi step process. Figure 4 shows a simplified schematic of the new database design and update procedure.

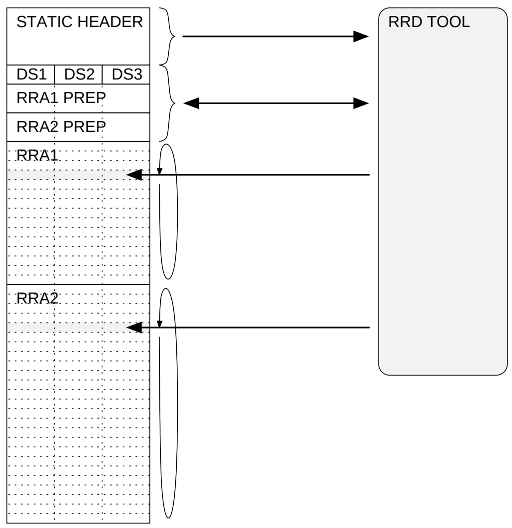

Figure 4: MRTG-3/RRD update procedure

An RRD can be configured to accept data from a number of data sources in parallel. A data source can be anything, be it an octet counter or the output of a temperature sensor. Each RRD operates at a configurable base time resolution. All data coming from the data sources is re-sampled at this resolution. The re-sampling of the data takes care of the problem that it is not always possible to get new data at the desired point in time but further processing and storage is much simpler when the data is equally spaced along the time axis. Figure 5 shows the re-sampling process for a counter type data source at a 300 second interval.

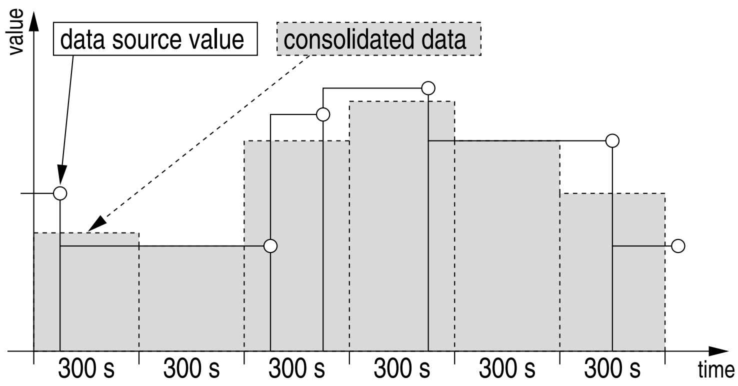

Figure 5: Data resampling process for a counter type data source at a 300 second interval

The basic idea behind improving logging performance was to reduce the amount of data which has to be transfered between memory and disk. This is achieved by storing data in a round robin manner into preallocated storage areas called Round Robin Archives (RRA) inside the RRD. Each Round Robin Archive has its special properties for time resolution, size and consolidation method. The update interval of an RRA must be a multiple of the base update interval of the RRD. Several values at the RRD's base resolution are consolidated into one value at the RRA's resolution using the consolidation method defined for this RRA. An array of pointers identifies the most current entry in each RRA, such that only one write operation is necessary to update an RRA.

One Round Robin Database (RRD) can contain any number of Round Robin Archives (RRA). For Example, one RRA could be configured to store data at the base resolution of the RRD for a few days, while another one stores the daily averages for 5 years. It is also possible to configure an RRD which mimics the data storage properties of an MRTG-2 logfile.

The time to update an RRD with new data values is roughly proportional to the number of Round Robin Archives it contains plus a constant part for reading the header portion of the RRD and time-aligning new data values.

To help guarantee data quality, the RRD format allows to specify validity conditions like the minimum update frequency required or the minimum and maximum values allowed for a data source. If a condition is not met, the data supplied is regarded invalid and an *unknown* data value is stored in the RRD.

The new design allows to store in the order of a thousand data values per second in a Round Robin Database. This rate drops dramatically if the RRD

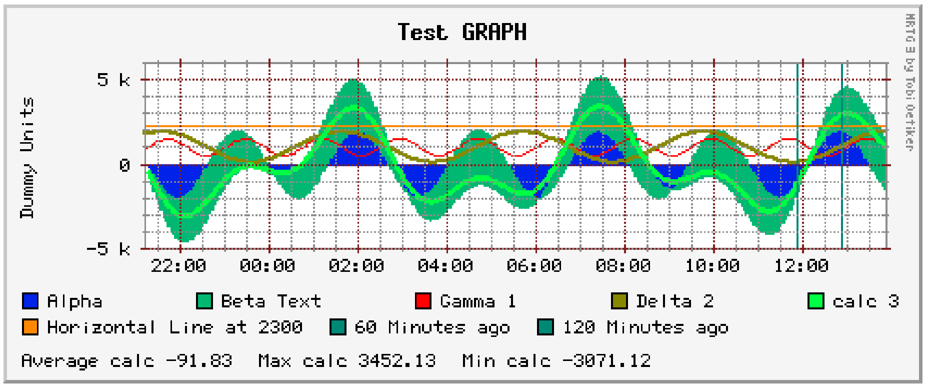

Figure 6: A sample graph from RRD Tool showing some of its features

file is accessed via NFS or if the disk cache of the machine is too small compared to the number of RRD files involved in the test. The potential NFS and cache memory problems aside, not much difference was seen between a Pentium 120 running Linux and a Sparc Ultra Enterprise 2 running Solaris at 200 MHz. A direct comparison with MRTG-2 is not possible because MRTG-2 integrates the graph creation into the data logging process.

Graph generation

MRTG-2 is focused on traffic graphs. Most parameters of these graphs are hard-coded. The graphing engine of the RRD Tool, however, is as flexible as the new RRD format. It allows to produce graphs of any size, spanning an arbitrary time period and to draw data from a number of data sources stored in different RRDs.

Whenever possible, the graphing engine determines sensible default values for its configurable parameters, allowing the user to concentrate on the fine tuning. Almost every aspect of the graph's visual appearance is configurable by overriding the automatic default values. Often configuration is not necessary, because the RRD Tool has several functions which automatically tune features like axis labels and scaling to fit the displayed data.

The graphing part of the RRD Tool also has some built-in analysis capability. It can calculate the maximum, minimum and average values from any data source. For more complex requirements, it is possible to use RPN math on any number of data sources and then graph the result. Figure 6 shows a sample graph demonstrating some of the capabilities of the RRD Tool.

Using the RRD Tool

The RRD Tool exists as a stand-alone program called *rrdtool*, which takes its instructions either from the command-line or from a pipe. The preferred way of using RRD Tool, though, is to access its functions directly through Perl bindings. This saves the overhead generated from executing a new *rrdtool* process for every operation, and, as opposed to attaching *rrdtool* via a set of pipes to a Perl script, it even works under Windows NT.

The following example shows how to set up a new Round Robin Database called `demo.rrd`.

```
rrdtool create demo.rrd --step=300 DS:COUNTER:400:0:1000000 \
  DS:GAUGE:600:-100:100 RRA:AVERAGE:1:1000 RRA:AVERAGE:10:2000 \
  RRA:MAX:10:2000
```

The `demo.rrd` database has a base update interval of 300 seconds. It accepts input from two data sources and stores its data in two Round Robin Archives. The first data source is a counter type which must be read at least every 400 seconds. It counts between 0 and 1,000,000 units per second. The second data source is a gauge type with a minimum read interval of 600 seconds. It outputs values between −100 and 100. Any values not complying with the limits defined for each data source will be recorded as *unknown*.

Data is stored in three Round Robin Archives. The first one stores the last 1,000 value sets in 300-second intervals (one base interval). The second RRA stores 2,000 value sets in 3,000-second intervals, building the average of 10 base intervals. Finally the third RRA has the same properties as the second, except that it stores the maximum 300-second values seen over the last 10 base intervals.

Storing data into an RRD is simple:

```
rrdtool update demo.rrd DATA:1994982:U
```

This updates the RRD with a reading from the first data source (the counter) and an *unknown* reading from the second data source. The time-stamp for this update is the current time. If desired, the time can be specified via the `--time` option.

After the RRD has been filled with some data, the graphing function of RRD Tool can be used to generate a visual representation of the data collected so far.

```
rrdtool graph demo.gif --start=-86400 --title="LISA Demo Graph" \
  --vertical-label='Degree Fahrenheit' \
  DEF:celsius=demo.rrd:1:AVERAGE \
  "CDEF:fahrenheit=celsius,9,*,5,/,32,+" \
  AREA:fahrenheit#ff0000:"Temperature in Room J97" \
  GPRINT:fahrenheit:AVERAGE:"Average for the last 24h %2.1fF"
```

The output created by this command is shown in Figure 7. It shows the temperature data from the last 24 hours (86400 seconds). The temperature data is expressed in degree Fahrenheit, is derived from the Celsius scale using RPN math on the `CDEF` line in the example above.

RRD Tool in the Real World

Otmar Lendl's (<`O.Lendl@Austria.EU.net`>) implementation of an in-house network monitoring solution based on RRD for EUnet Austria is an example for RRD Tool being used in a productive environment.

In Lendl's setup RRD Tool is used to monitor and graph about 6000 variables from 1200 network interfaces, servers and dial-in lines in 5 minute intervals. Their system runs on a low-end SPARCstation-5/170 at a load of about 0.2. The

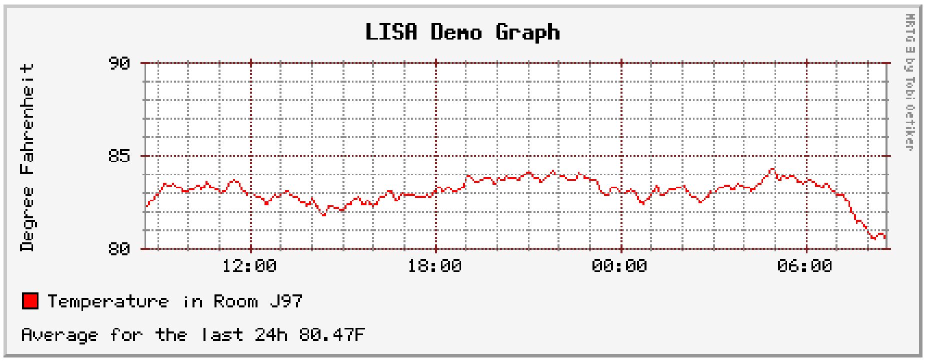

Figure 7: Output of the sample line

data-acquisition software is written in Perl 5 using the Perl-bindings for RRD and UCD SNMP. It is split into a scheduler (using the EventServer Perl module) and multiple worker processes which communicate via UDP. The visualization is implemented as *mod_perl* scripts running under Apache 2.3. All images, as well as most of the HTML, are generated dynamically.

Unfortunately, the web pages generated by this is setup are not accessible from outside of EuNet Austria for privacy reasons.

Additional plans for MRTG-3

SNMP data gathering

The Round Robin Database moves the performance bottleneck of MRTG to the data gathering component. The plan for improving SNMP data gathering performance is to issue several SNMP requests in parallel. This works around network latency as well as problems with routers that answer SNMP requests slowly.

Graphs on demand

Because the generation of graphs is quite expensive, it is not sensible to update thousands of GIF images on a regular basis. It is more efficient to generate the graphs when a user wants to see them. The graph shown in Figure 6 took about 0.3 second to generate on a Pentium 120. This means that graphs can be created on the fly and still an acceptable response time can be achieved. For high traffic sites this could be coupled to a graph cache so that the each graph is only regenerated when it is out of date.

HTML generation

In MRTG-2 the look of the generated HTML pages was tuned using a large number of configuration options. MRTG-3 will work with template files and therefore make the design of HTML pages both simpler and more flexible.

Configuration

While MRTG-2 was a monolithic program, version 3 will be a set of Perl modules which can be assembled into custom monitoring applications. The user can decide which modules to use.

One module will provide a high-level user interface for making MRTG-2-like applications. Scripts which `use` this module will consist of two parts: In the first part, the user will define all the data sources to monitor. In the second part, an event handler will be called which fetches the requested data and updates the respective RRDs and HTML pages in an optimized order.

Summary

MRTG 1.0 was conceived as an in-house application. Its publication on the Internet showed that there was a considerable demand for such a program. Many free tools like NeTraMet[1], Scotty[6], CMU SNMP were available for retrieving data on the current state of a network link, but MRTG's approach for long term analysis and the friendly presentation on the Web was new. Many users stated that they were able to monitor network links with MRTG "better" than with any commercial tool available at the time.

Today, the NetSCARF project's *Scion*[7] is providing a solution somewhat similar to MRTG. *BigBrother*[3] and *CFlowd*[2] tools applied in the same area.

The RRD package dramatically improves the logging performance and has better configurability than the MRTG-2 software. This makes it suitable for a broader range of applications, both in the area where a lot of data has to be gathered as well as in cases that call for more complex monitoring configurations than a simple two parameter traffic graph.

Acknowledgments

MRTG is an application which only exists due to the participation of many people. I would like to thank the following people and institutions in particular: Dave Rand for initiating MRTG-2 development with his *rateup* utility; Simon Leinen for the SNMP library written entirely in Perl; Thomas Boutell for the gd library which is used to generate the graphical output of MRTG; the De Montfort University in Leicester, UK and the Department of Electrical Engineering of the Swiss Federal Institute of Technology in Zurich for supporting the MRTG development; and last but not least to Larry Wall, the creator of Perl.

Availability and Support

MRTG-2 is the current release. It is available from the MRTG home page on `http://ee-staff.ethz.ch/~oetiker/webtools/`. All the code available in connection with MRTG-3 and RRD Tool development is available from `http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/3.0/`. MRTG is available under the terms of the GNU GPL. For exchange with other MRTG users there are mailing lists available. For commercial support please contact the author. Further information can be found on the MRTG home page.

Author Biography

Tobias Oetiker got a Master's degree in Electrical Engineering from the Swiss Federal Institute of Technology, Zurich (ETHZ) in 1995. After working for one year at De Montfort University in Leicester, UK doing Unix system management, he returned to Switzerland and has since been employed by the Department of Electrical Engineering of the Swiss Federal Institute of Technology as a toolsmith and system manager.

References

[1] Nevil Brownlee. **NeTraMet**, a network traffic accounting meter for PC and UNIX. `http://www.auckland.ac.nz/net/Accounting/`

[2] Daniel W. McRobb and John Hawkinson. **Cflowd**, an experimental software to collect data from Cisco's flow-export feature. `http://engr.ans.net/cflowd/`

[3] Sean MacGuire. **Big Brother**, a tool for proactive network monitoring. `http://www.iti.qc.ca/users/sean/bb-dnld/`

[4] Simon Leinen. **Perl 5 SNMP Module**, an SNMP client implemented entirely in Perl. `http://www.switch.ch/misc/leinen/snmp/perl/`

[5] Thomas Boutell. **GD Lib**, a graphics library for fast creation of GIF images. `http://www.boutell.com/gd/`

[6] Jürgen Schönwälder. **Scotty**, a Tcl Extensions for Network Management.`http://www.ibr.cs.tu-bs.de/projects/scotty/`

[7] W. Norton and A. Adams. **Scion**, a tool to query SNMP-aware network equipment for performance information, and make that information available on the Web. `http://www.merit.edu/net-research/netscarf/`

[8] Jon Kay. **Internet Measurement Tool Survey** `http://www.caida.org/Tools/taxonomy.html`