

RRDtool advanced Topics

Tobias Oetiker

OpenNMS User Conference 2013

1 A different kind of Database

creating a simple rrd

```
1 #!/bin/sh
2 PATH=/scratch/rrd4/bin:$PATH
3 R=rrdtool
4 $R create first.rrd \
5     --step=300 \
6     --start=1199999699 \
7     DS:temperature:GAUGE:600:-40:100 \
8     RRA:AVERAGE:0.4:1:5 \
9     RRA:AVERAGE:0.4:3:2 \
10    RRA:MIN:0.4:3:2 \
11    RRA:MAX:0.4:3:2
```

One Datasource, 4 Round Robin Archives

feeding data

```
1
2 #!/bin/sh
3 R=rrdtool
4 u() {
5     $R update first.rrd $1
6 }
7
8 u 1199999700:00
9 u 1200000000:10
```

Feed in some data. One or several updates at once.

inside the database

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE rrd SYSTEM
3     "http://oss.oetiker.ch/rrdtool/rrdtool.dtd">
4 <rrd> <version> 0003 </version>
5     <step> 300 </step> <!-- Seconds -->
6     <lastupdate> 1200000900 </lastupdate>
7     <!-- 2008-01-10 22:35:00 CET -->
8
9     <ds>
10        <name> temperature </name>
11        <type> GAUGE </type>
12        <minimal_heartbeat> 600 </minimal_heartbeat>
13        <min> -4.0000000000e+01 </min>
14        <max> 1.0000000000e+02 </max>
15
16        <!-- PDP Status -->
17        <last_ds> 40 </last_ds>
18        <value> 0.0000000000e+00 </value>
19        <unknown_sec> 0 </unknown_sec>
20    </ds>
21
22
23
24    <!-- RRA:AVERAGE:0.4:1:5 -->
25    <rra>
```

```

26     <cf> AVERAGE </cf>
27     <pdp_per_row> 1 </pdp_per_row> <!-- 300 seconds -->
28
29     <params>
30     <xff> 4.0000000000e-01 </xff>
31     </params>
32     <cdp_prep>
33     <ds>
34     <primary_value> 4.0000000000e+01 </primary_value>
35     <secondary_value> 0.0000000000e+00 </secondary_value>
36     <value> NaN </value>
37     <unknown_datapoints> 0 </unknown_datapoints>
38     </ds>
39     </cdp_prep>
40     <database>
41     <row><v> NaN </v></row>
42     <row><v> 1.0000000000e+01 </v></row>
43     <row><v> 2.0000000000e+01 </v></row>
44     <row><v> 3.0000000000e+01 </v></row>
45     <row><v> 4.0000000000e+01 </v></row>
46     </database>
47 </rra>
48
49 <!-- RRA:AVERAGE:0.4:3:2 -->
50 <rra>
51     <cf> AVERAGE </cf>
52     <pdp_per_row> 3 </pdp_per_row> <!-- 900 seconds -->
53
54     <params>
55     <xff> 4.0000000000e-01 </xff>
56     </params>
57     <cdp_prep>
58     <ds>
59     <primary_value> 2.0000000000e+01 </primary_value>
60     <secondary_value> 3.0000000000e+01 </secondary_value>
61     <value> 4.0000000000e+01 </value>
62     <unknown_datapoints> 0 </unknown_datapoints>
63     </ds>
64     </cdp_prep>
65     <database>
66     <row><v> NaN </v></row>
67     <row><v> 2.0000000000e+01 </v></row>
68     </database>
69 </rra>
70
71
72
73 <!-- RRA:MIN:0.4:3:2 -->
74 <rra>
75     <cf> MIN </cf>
76     <pdp_per_row> 3 </pdp_per_row> <!-- 900 seconds -->
77
78     <params>
79     <xff> 4.0000000000e-01 </xff>
80     </params>
81     <cdp_prep>
82     <ds>
83     <primary_value> 1.0000000000e+01 </primary_value>
84     <secondary_value> 3.0000000000e+01 </secondary_value>
85     <value> 3.0000000000e+01 </value>
86     <unknown_datapoints> 0 </unknown_datapoints>
87     </ds>
88     </cdp_prep>
89     <database>
90     <row><v> NaN </v></row>
91     <row><v> 1.0000000000e+01 </v></row>
92     </database>
93 </rra>
94

```

```

95
96 <!-- RRA:MAX:0.4:3:2 -->
97 <rra>
98 <cf> MAX </cf>
99 <pdp_per_row> 3 </pdp_per_row> <!-- 900 seconds -->
100
101 <params>
102 <xff> 4.0000000000e-01 </xff>
103 </params>
104 <cdp_prep>
105 <ds>
106 <primary_value> 3.0000000000e+01 </primary_value>
107 <secondary_value> 3.0000000000e+01 </secondary_value>
108 <value> 4.0000000000e+01 </value>
109 <unknown_datapoints> 0 </unknown_datapoints>
110 </ds>
111 </cdp_prep>
112 <database>
113 <row><v> NaN </v></row>
114 <row><v> 3.0000000000e+01 </v></row>
115 </database>
116 </rra>
117
118 </rrd>

```

The xml dump of the rrd file shows an approximation of the on-disk structure of the database. The rra database sections are re-ordered, so that they are in chronological order with the oldest at the top. Also the cdp sections are stored right after the header. The idea behind this design is that data that get written on every update is as close together as possible.

rrd features

- optimized for time-series data
- fixed size rotating data store
- constant on-disk size
- no maintenance
- on the fly consolidation

on-disk structure

```

+-----+
| Static Header                | RRD cookie, DB cfg
+-----+
: Data Source (DS) Definitions :
+-----+
: RR Archive (RRA) Definitions :
+=====+
| Live Head                    | last update time
+-----+

```

```

: PDP Prep per DS           : last value for diff
|-----|
: CDP Prep per RRA and DS   : intermediate storage
|-----|
: RRA Pointers              :
|=====|
: Round Robin Archives (RRA) :
+-----+

```

irregular data arrival intervals

```

1 #!/bin/sh
2 PATH=/scratch/rrd4/bin:$PATH
3 R=rrdtool
4 $R create real.rrd \
5     --step=300 \
6     --start=1199999699 \
7     DS:distance:COUNTER:600:-40:100 \
8     RRA:AVERAGE:0.4:1:5
9
10 u() {
11     $R update real.rrd $1
12 }
13
14 u 1200000000:0
15 u 1200000150:15
16 u 1200000310:31
17 u 1200000640:64
18 u 1200000910:91

```

To try things out lets assume that data arrives at irregular intervals. This is counter data. By synchronizing the data values with the arrival time we should get a constant rate stored in the database.

database after the irregular updates

```

1 $R fetch real.rrd -s 1200000000 -e 1200000899 AVERAGE
2
3           distance
4
5 1200000300: 1.0000000000e-01
6 1200000600: 1.0000000000e-01
7 1200000900: 1.0000000000e-01

```

- rrdtool re-binning at work
- major difference to a normal db
- all bins contain 1.0
- the time is the 'end-time' of the bin.

optimizing your rrrds

- update of multi DS RRD is cheap
- single update interval per RRD
- RRD modification is expensive
- RRD size and update performance are independent
- RRA complexity affects update performance

As long as your system is small (a few hundred RRDs) you should optimize for convenience. Only keep DSes together in one RRD that are tightly bound. For everything else create separate RRDs.

fetching data

fetch is for reading data from an rrd

```
1 RRA:AVERAGE:0.5:1:2 \  
2 RRA:AVERAGE:0.5:2:3
```

- one RRA with two 300s entries
- one RRA with three 600s entries

playing catch with fetch

first pull 300 seconds

```
> rrdtool fetch x.rrd -r 300 \  
-s 1200000600 -e 1200000900 AVERAGE
```

```
1200000900: 4.0000000000e+01  
1200001200: 5.0000000000e+01
```

then pull 900 seconds

```
> rrdtool fetch x.rrd -r300 \  
-s 1200000000 -e 1200000900 AVERAGE
```

```
1200000600: 2.5000000000e+01  
1200001200: 4.5000000000e+01
```

fetch recap

- looking for complete coverage
- resolution is only a suggestion
- time stamp in output marks the END of the period
- end-time differences caused problems
- since 1.3, only the start-time is relevant for coverage
- outside the rra everything is NaN

2 Graphing

rrdgraph syntax 101

for graph command syntax, there are two basic rules:

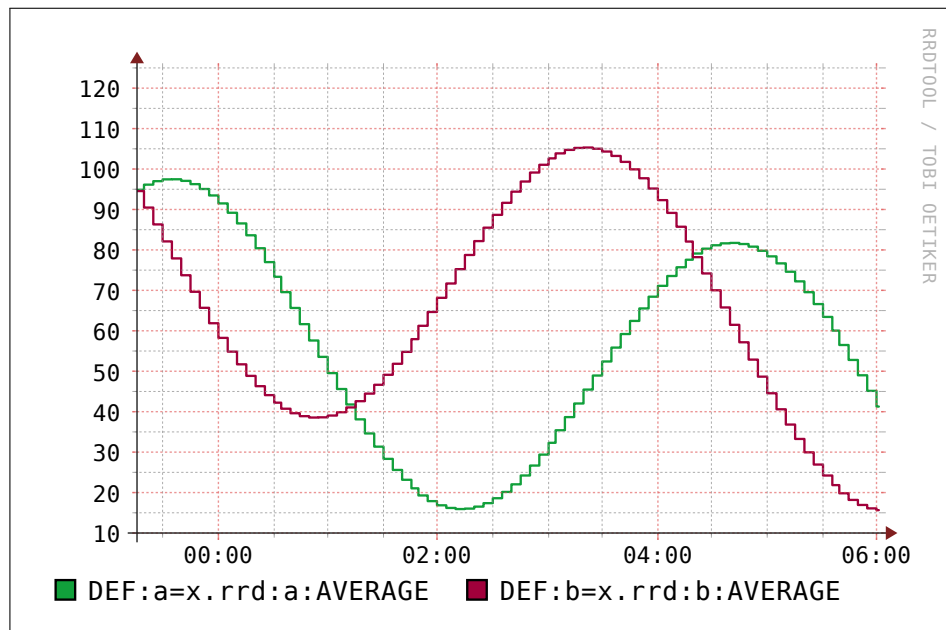
1. `--options` start with a double dash
2. graphing instructions start with a letter

```
rrdtool graph output
DEF:var=rrd:DS:AVERAGE
LINE:var#hex-rgb-color:Comment
```

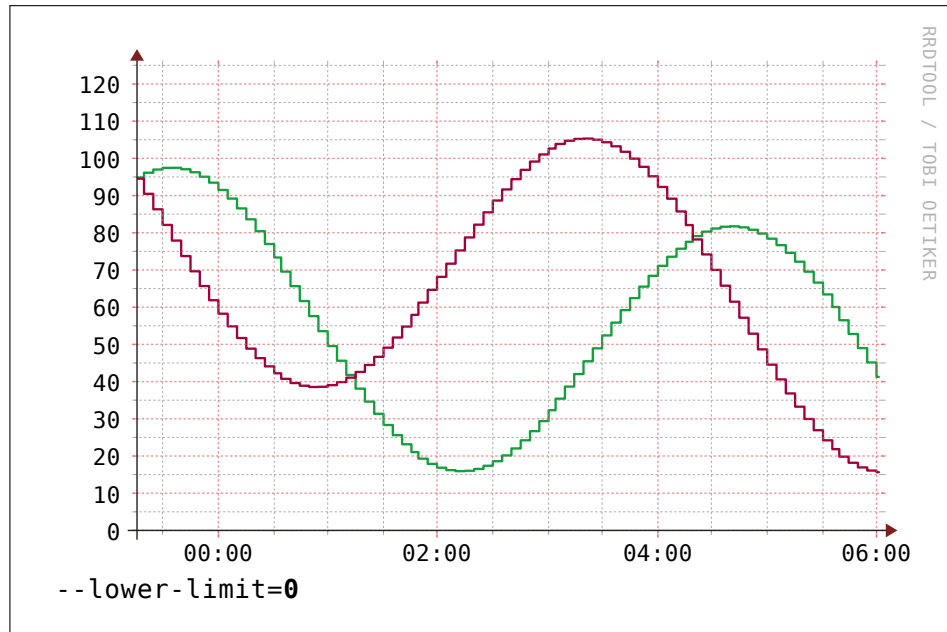
DEF and LINE are *graphing instructions*.

The `rrd graph` command is the most versatile of all `rrdtool` commands. It comes with its own little language, optimized for drawing graphs. There are two kinds of arguments. The options which start with a double-dash and the graphing instructions that start with an uppercase letter.

normal line

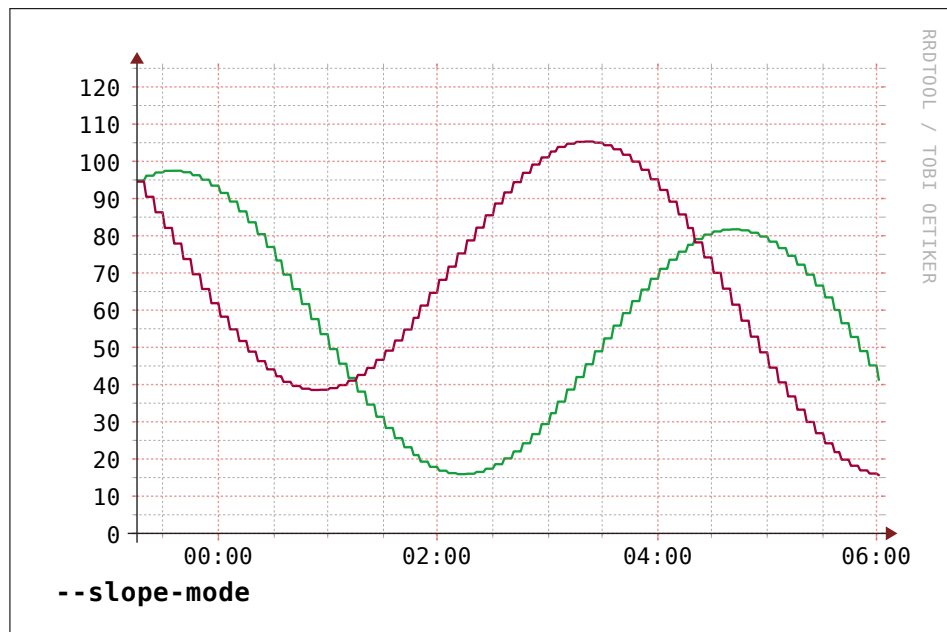


lower limit



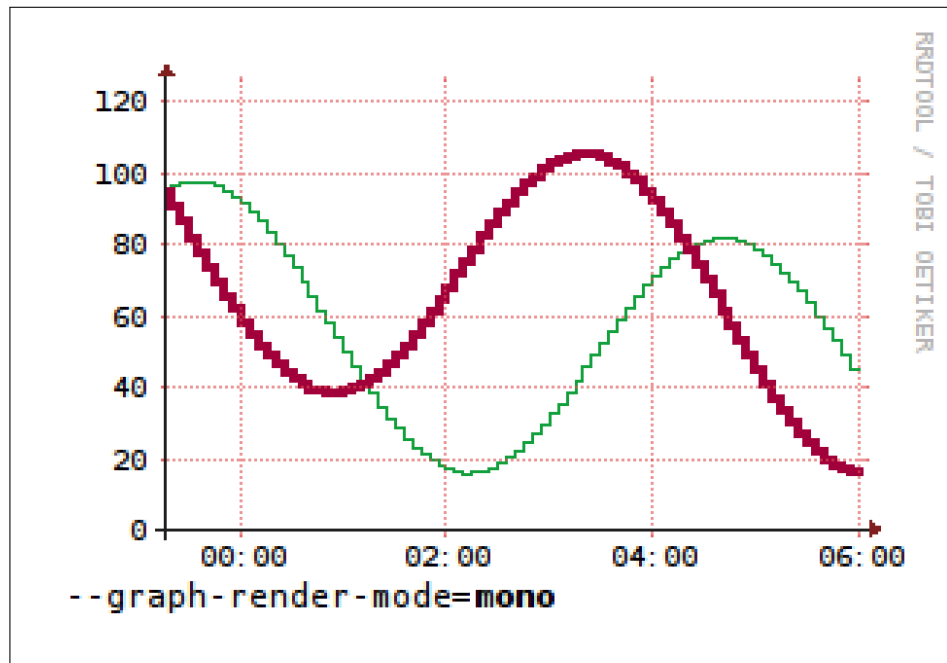
Unless you are a banker and are you drawing stock diagrams, make sure your graph displays the zero-y-value. Otherwise it is pretty difficult to judge the meaning of the graph since perspective is limited to the numbers on the y-axis.

slope mode

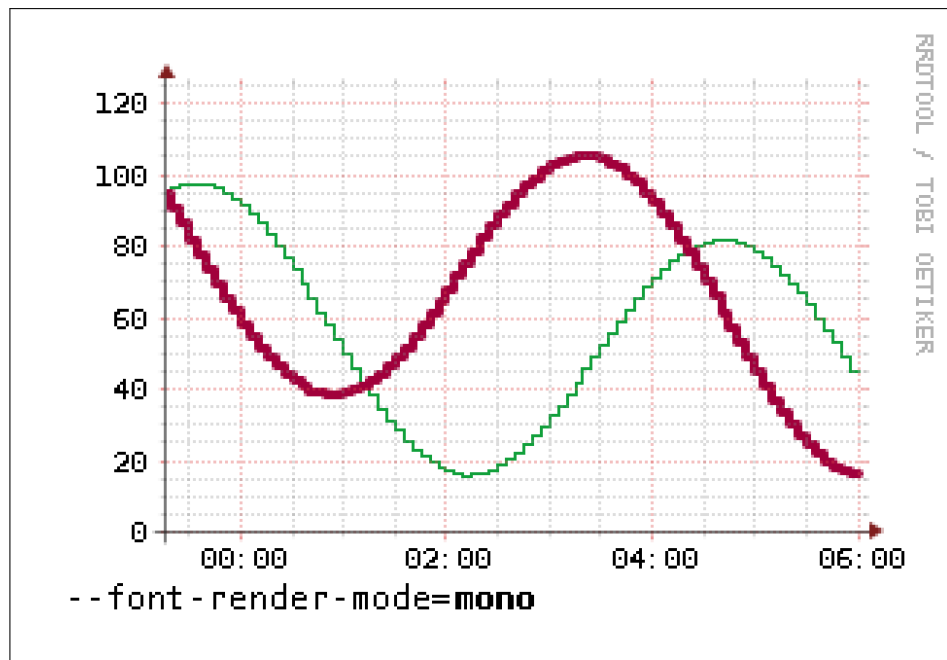


RRD graphs are pretty blocky. This is on purpose, since the data is blocky too. The slope mode is a little concession. It tilts the vertical connections between the 'blocks' by one pixel.

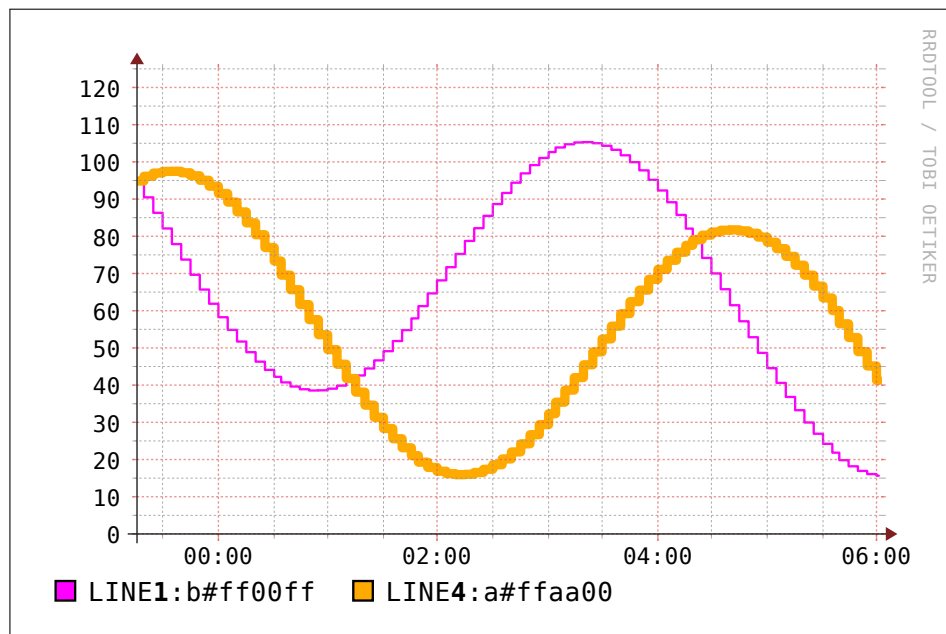
anti-anti-aliasing: graph



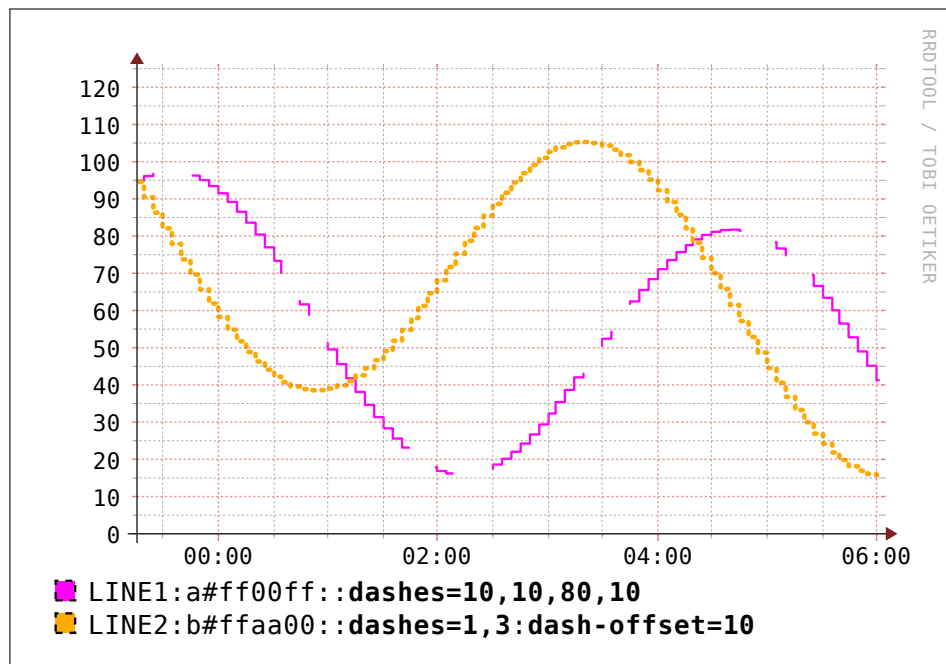
anti-anti-aliasing: font



line width

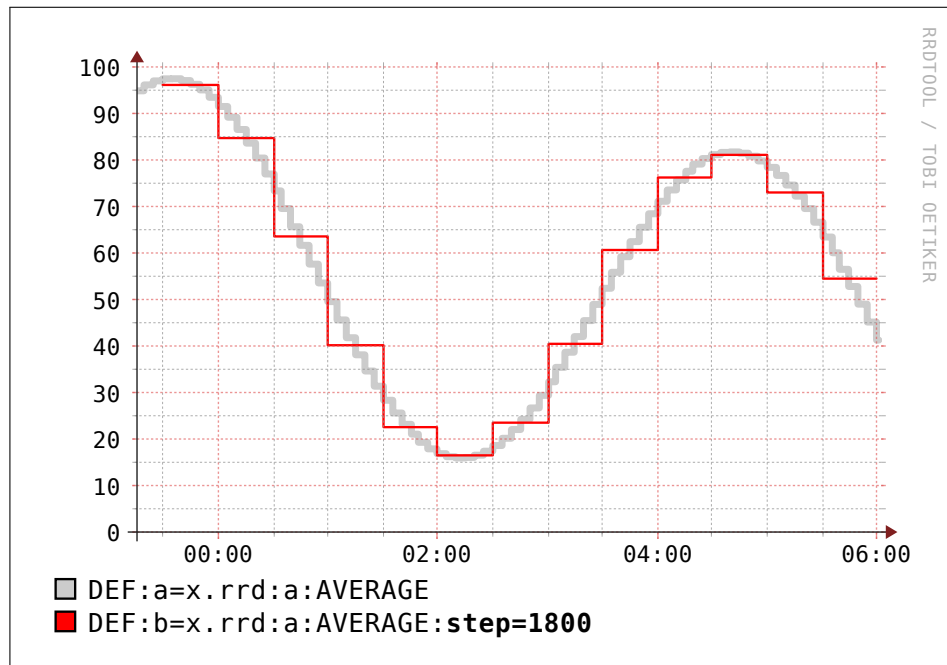


dashed line

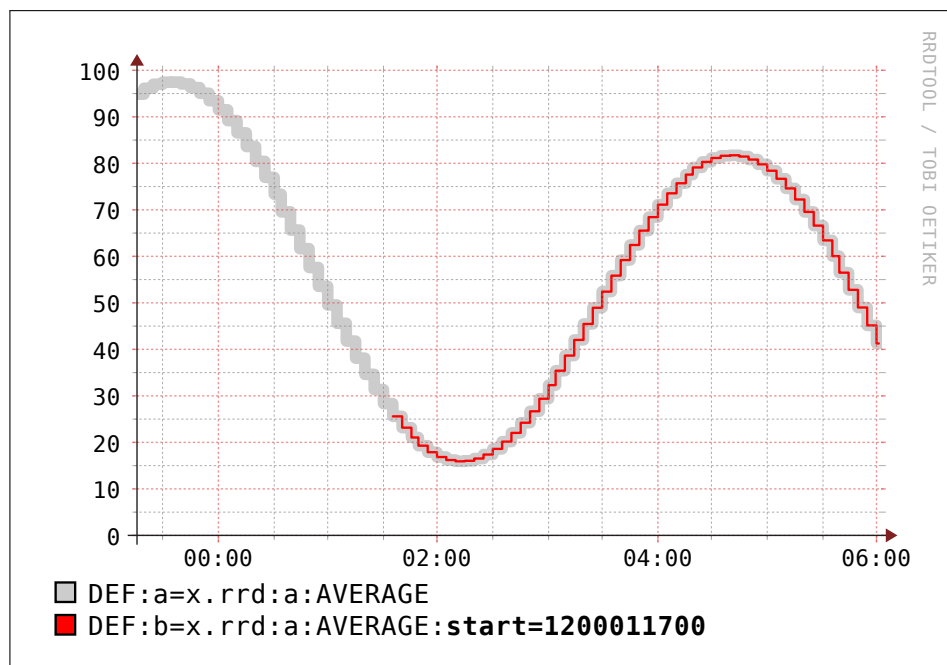


The numbers are an ON-OFF-ON-OFF-... pattern. The dash-offset property lets you shift the dashing of the line to the right.

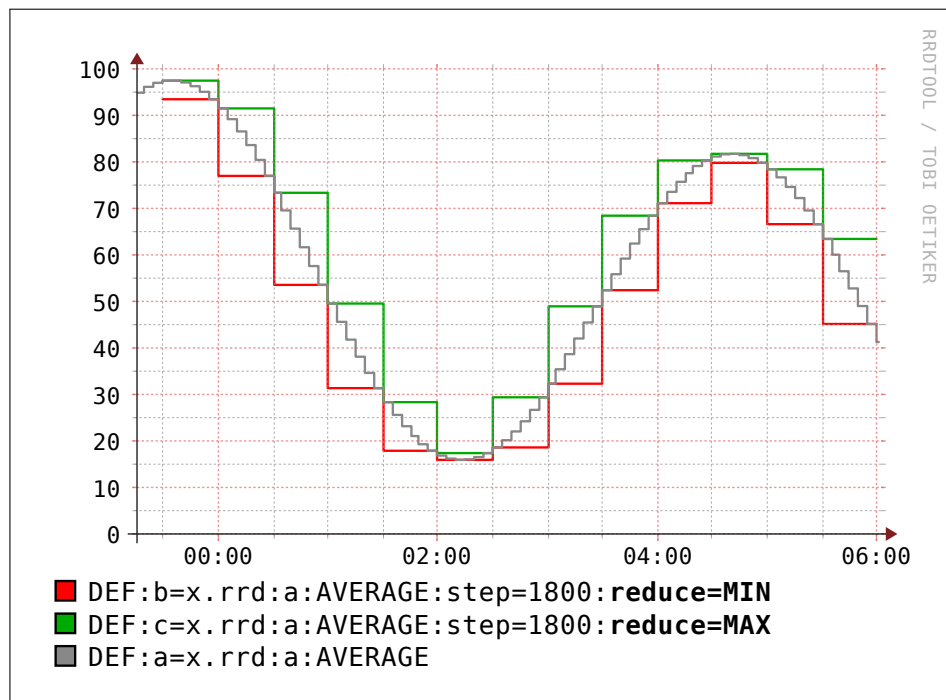
DEF with :step



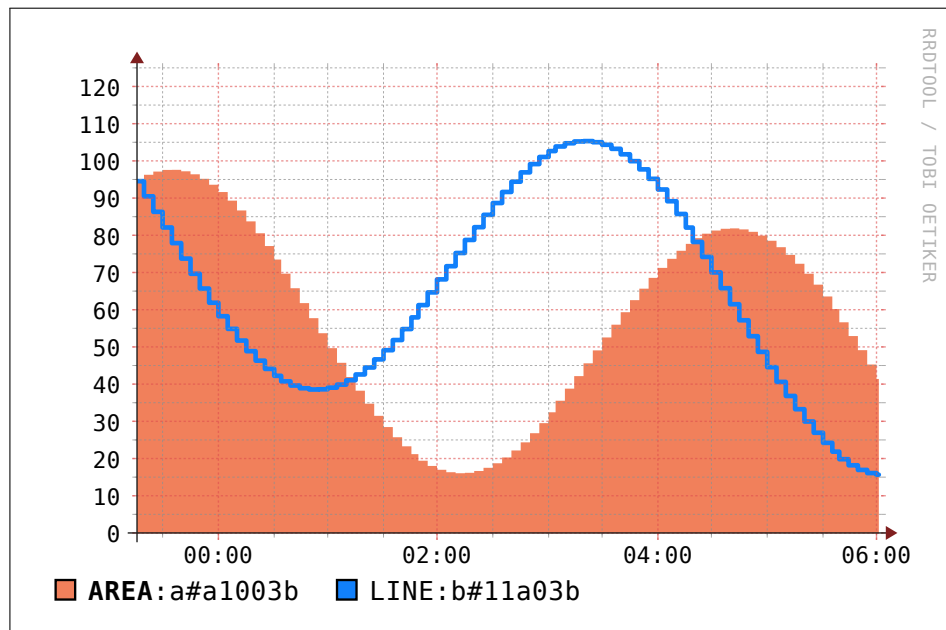
DEF with :start



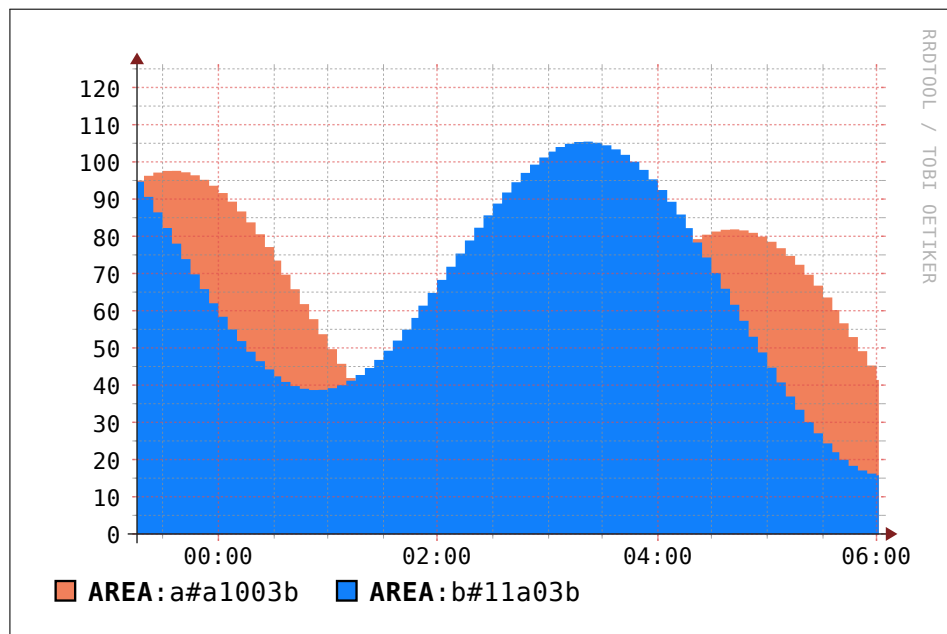
DEF with :reduce



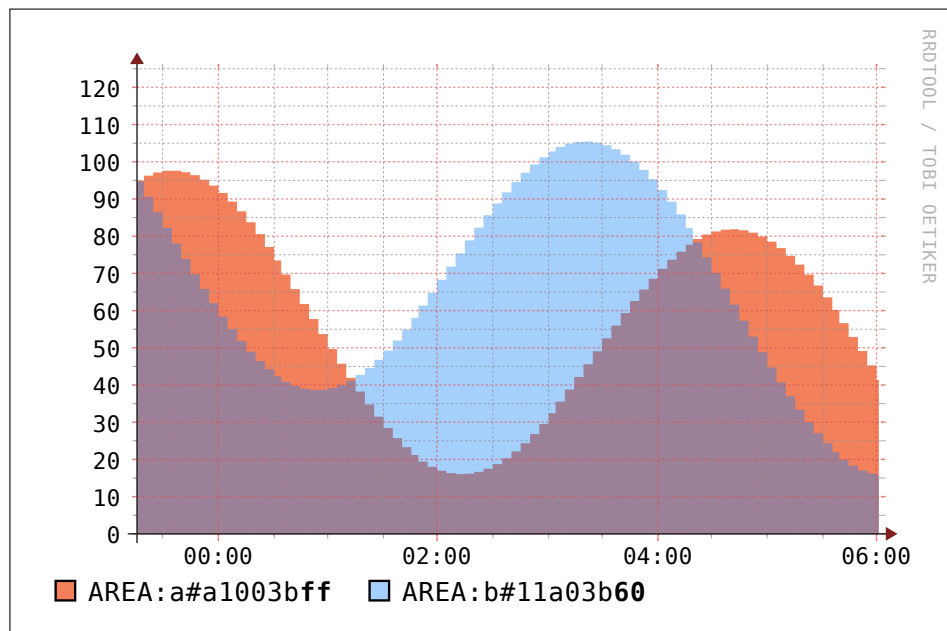
AREA simple



two AREAs

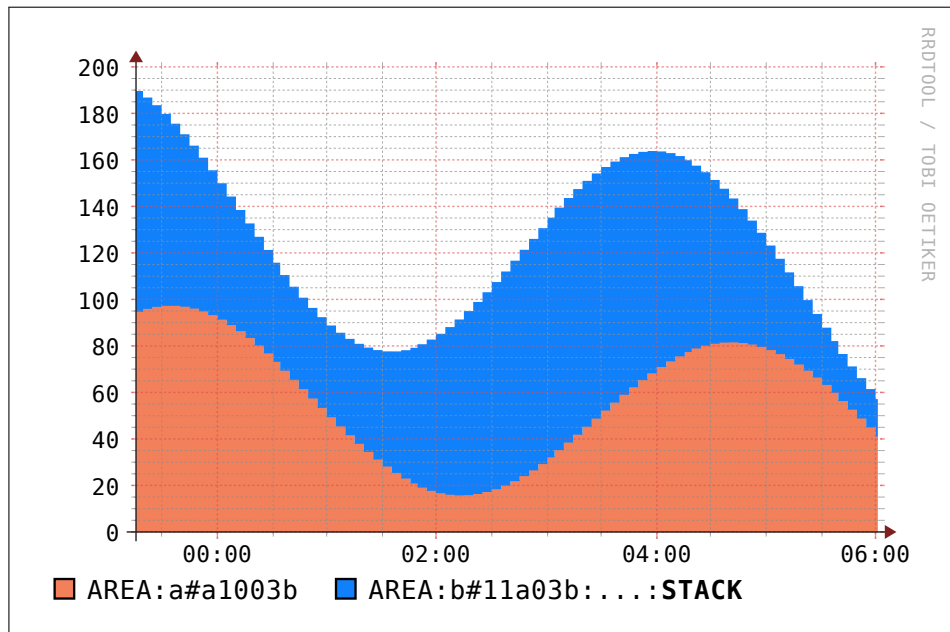


transparent AREA

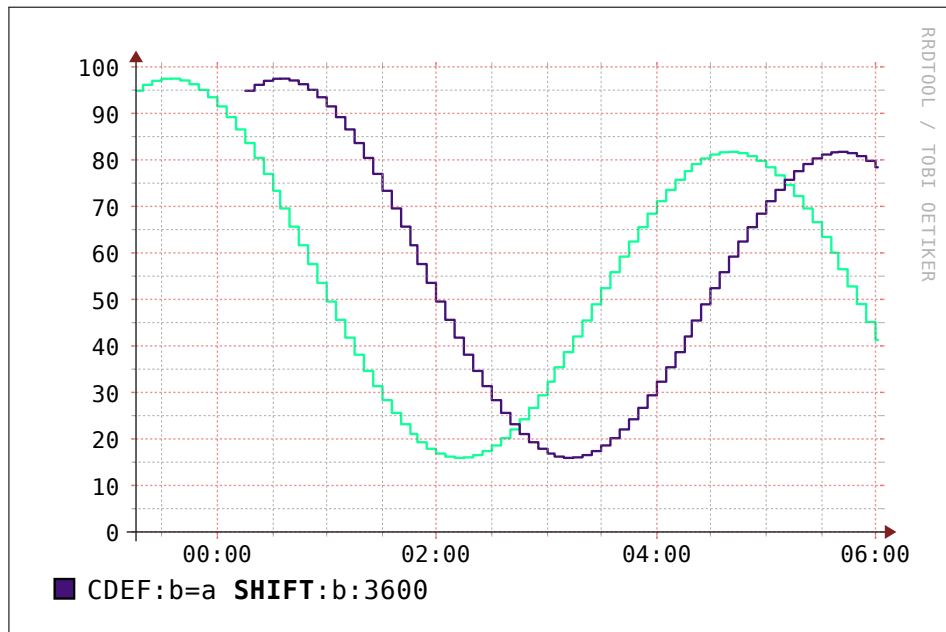


RRDtool creates real alpha transparency. You can set the whole graph to be transparent by setting the graph CANVAS and BACKGROUND colors to transparent.

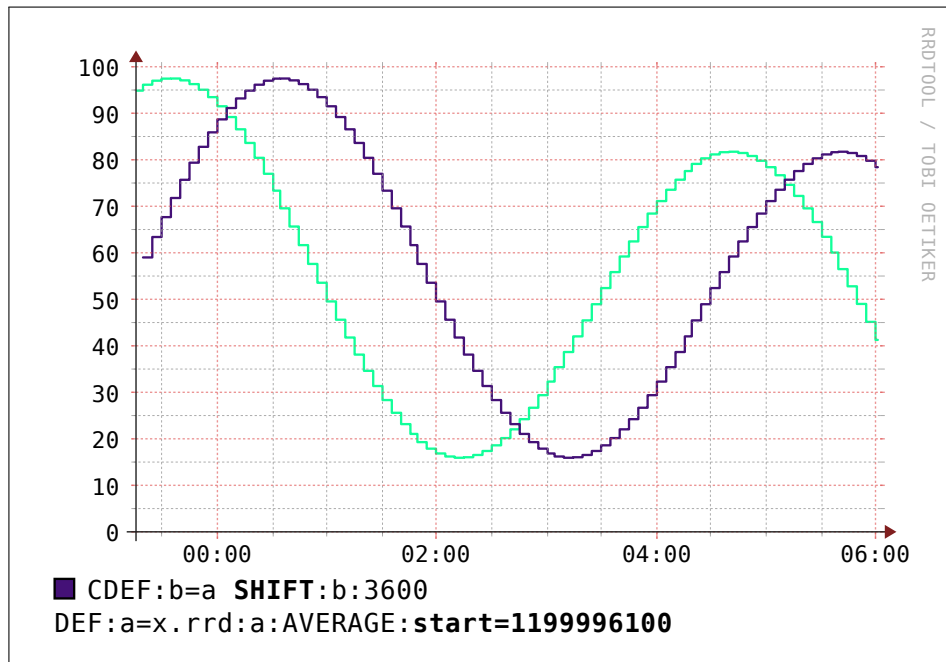
stacked AREA



time shift



shifting with extra data



A normal DEF line requests exactly as much data as it requires for drawing the graph. If you SHIFT the data, you may want to adjust the data fetched accordingly.

3 Revers Polish Notation (RPN) Math

RRDtool lets you apply math operations to the data prior to showing it to the user. It uses RPN math for this. If you ever owned a classic HP calculator, you may still remember how RPN math works. For all the others there is a little example below, that shows how to do a little addition in RPN.

RPN basics: Step 0

$$15 + 23 = 38$$

1: NAN
2: NAN
3: NAN

RPN basics: Step 1

$$15 + 23 = 38$$

[15] 1: **15**
 2: NAN
 3: NAN

RPN basics: Step 2

$$15 + \mathbf{23} = 38$$

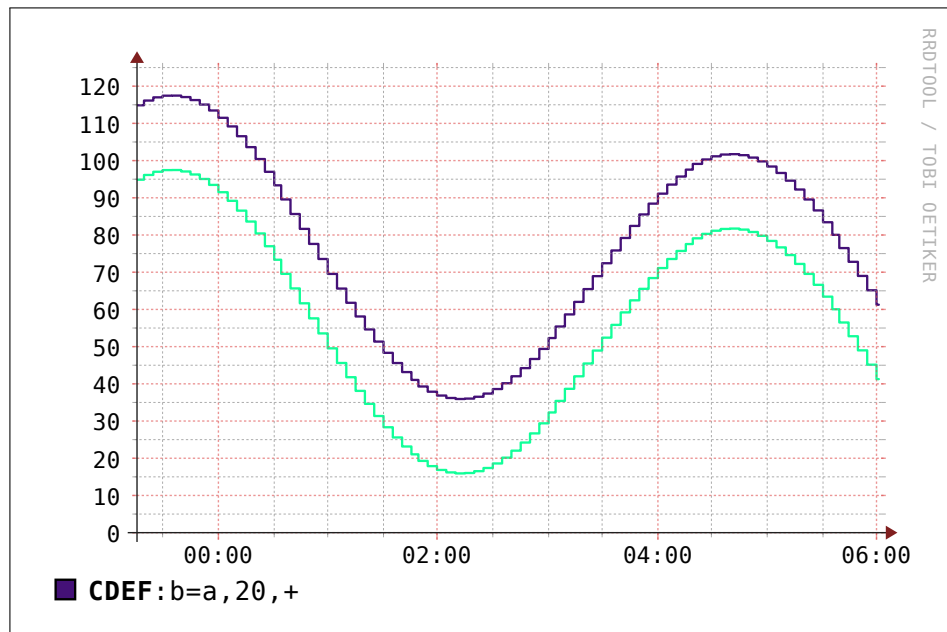
[23] 1: **23**
 2: 15
 3: NAN

RPN basics: Step 3

$$15+23 = 38$$

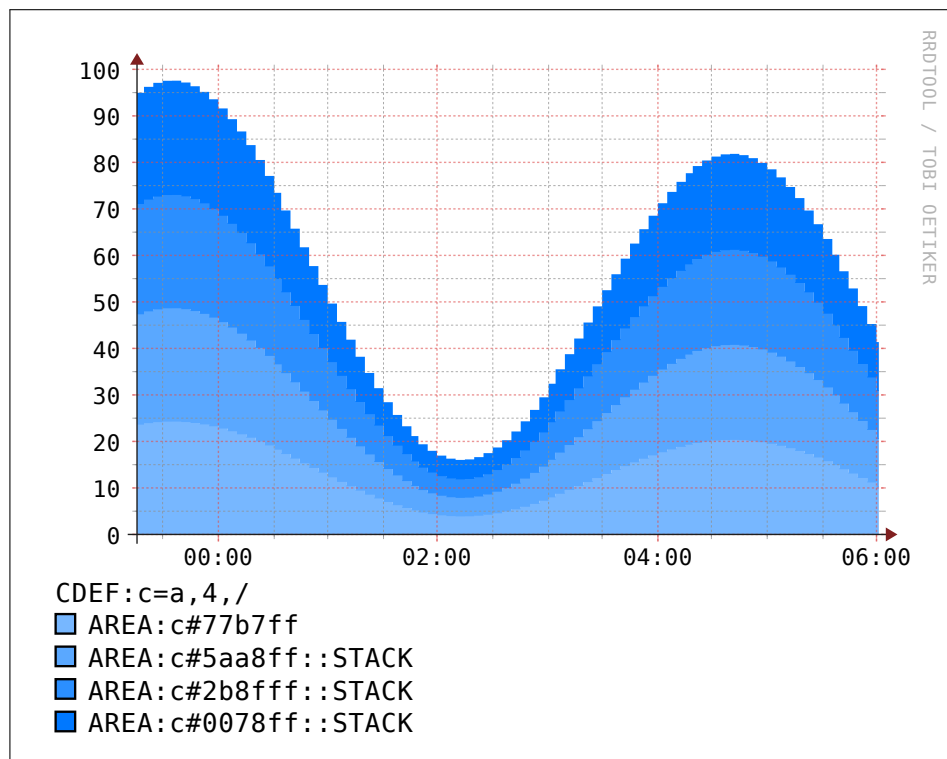
[+] 1: **38**
 2: NAN
 3: NAN

math in the graph (+)

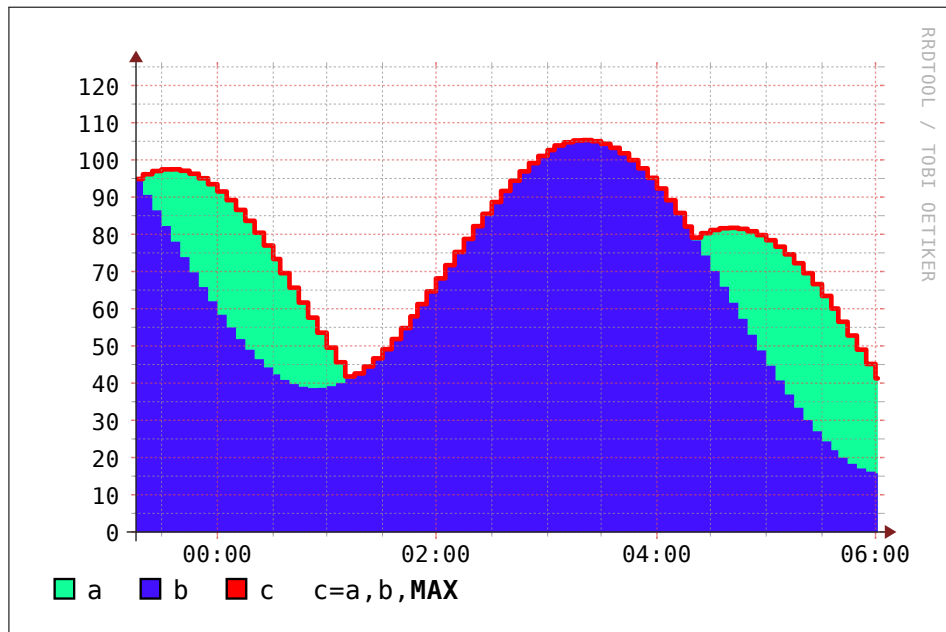


A simple addition. We add a fixed value to a data source. Note that at least one data source must appear inside a CDEF expression. The input to a CDEF expression can come from another CDEF expression.

simple gradient

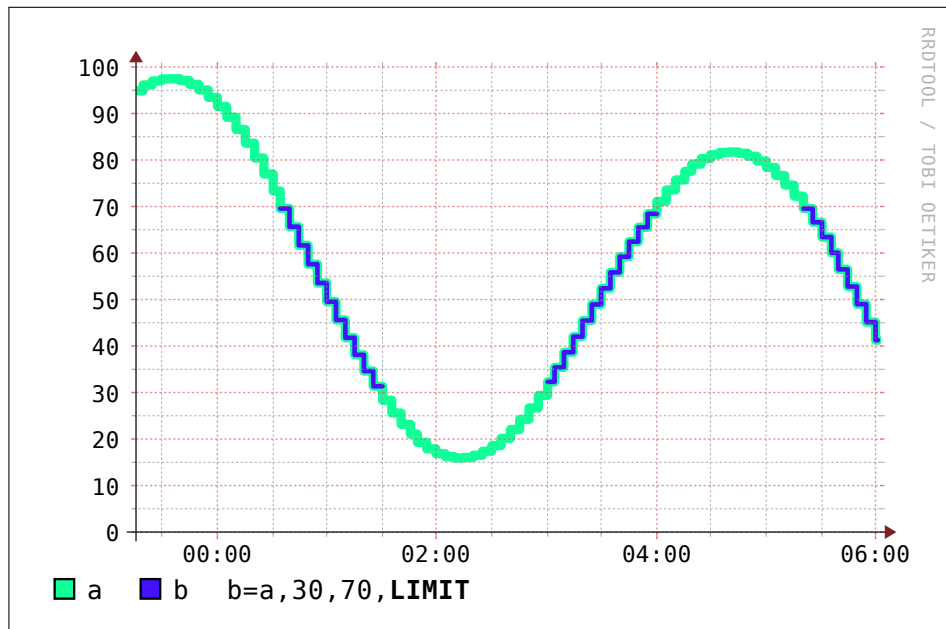


the MAX function



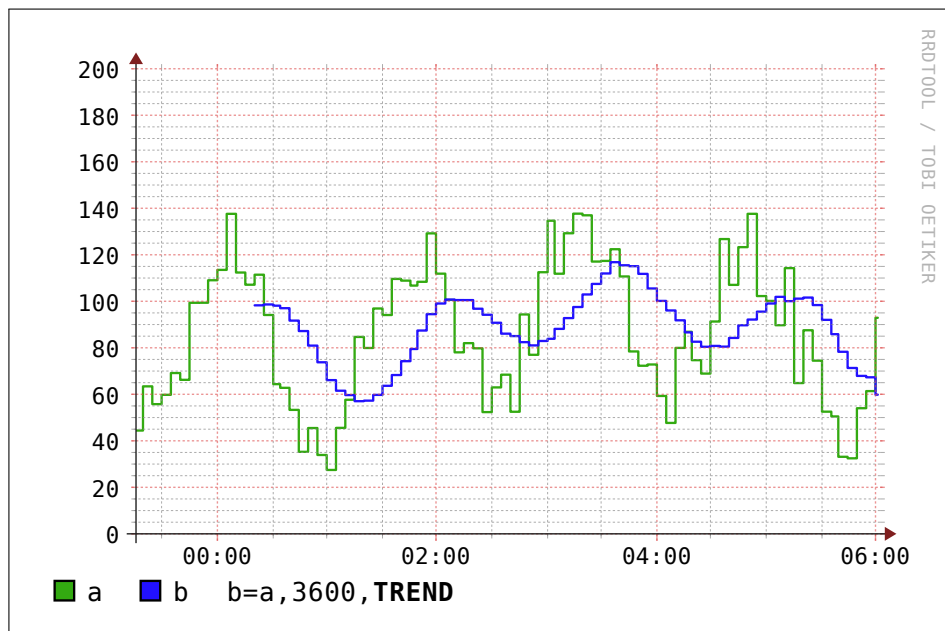
The MAX function operates on two values. In this example the input comes from two different data sources.

the LIMIT function



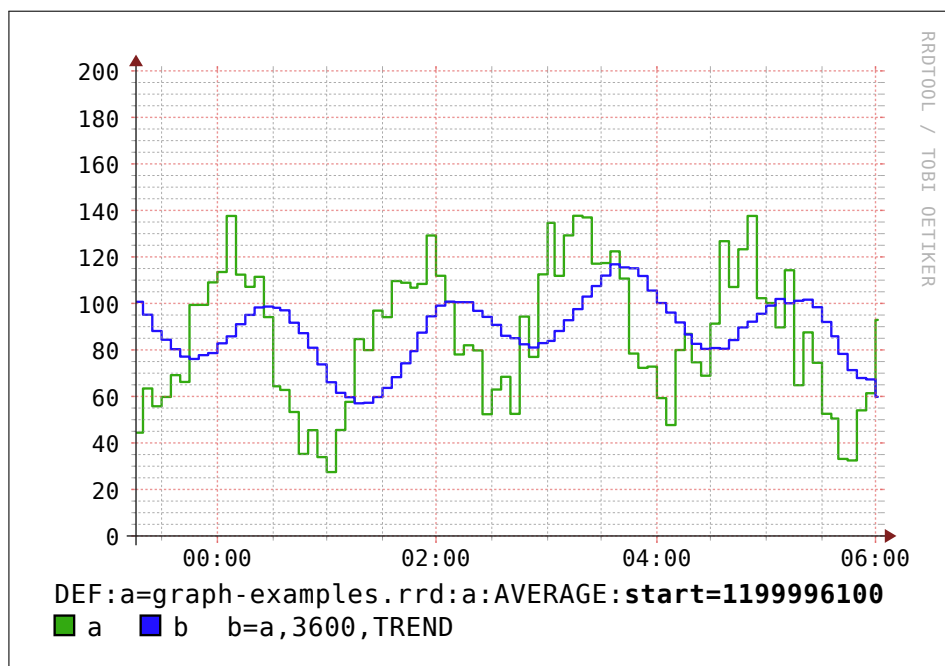
The LIMIT function will return UNKNOWN as soon as the input value is outside the given range. UNKNOWN data does not get drawn.

the TREND function



If a data source varies massively, the TREND function lets you smooth away by building a moving average. By calculating the average, the output gets shifted by the length of the TREND calculation.

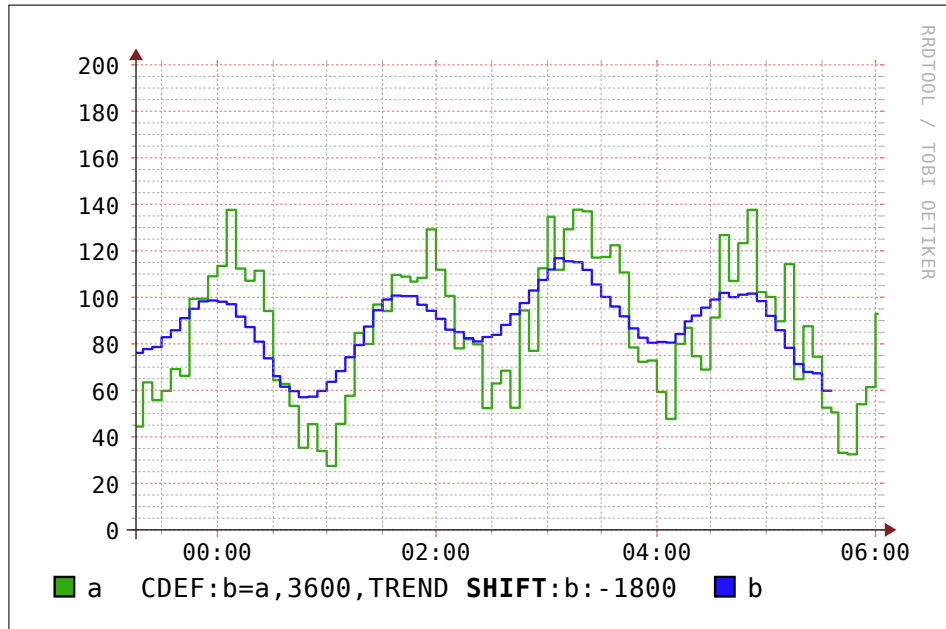
the TREND with early start



In the previous graph there was a bit of data missing at the left border of the graph. This was because rrdgraph loads exactly the amount of data that is required in the

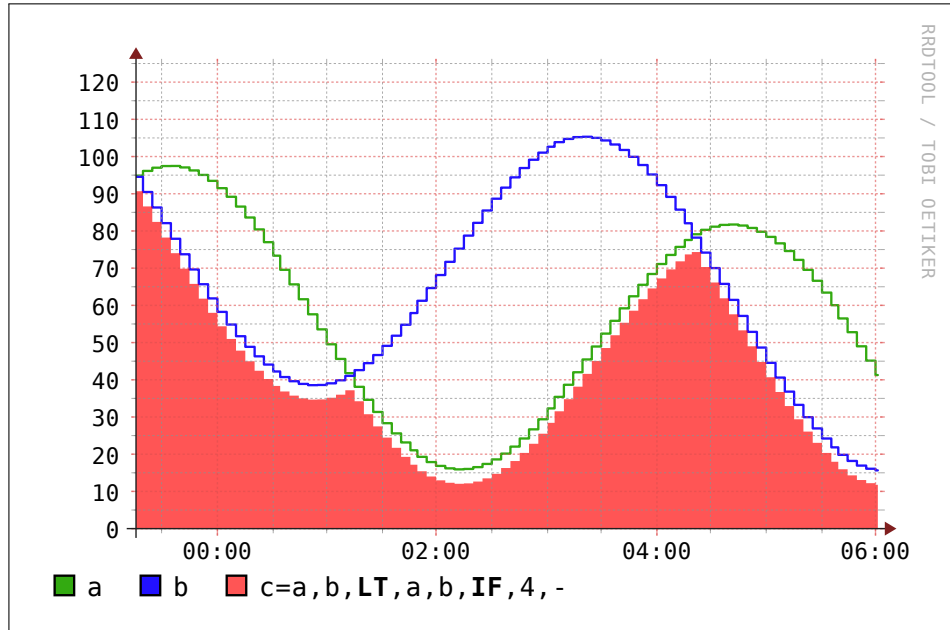
graph (yes same story as before). By loading more data, we can provide the TREND function with enough input, so that it can calculate the first few pixels as well.

the TREND and SHIFT



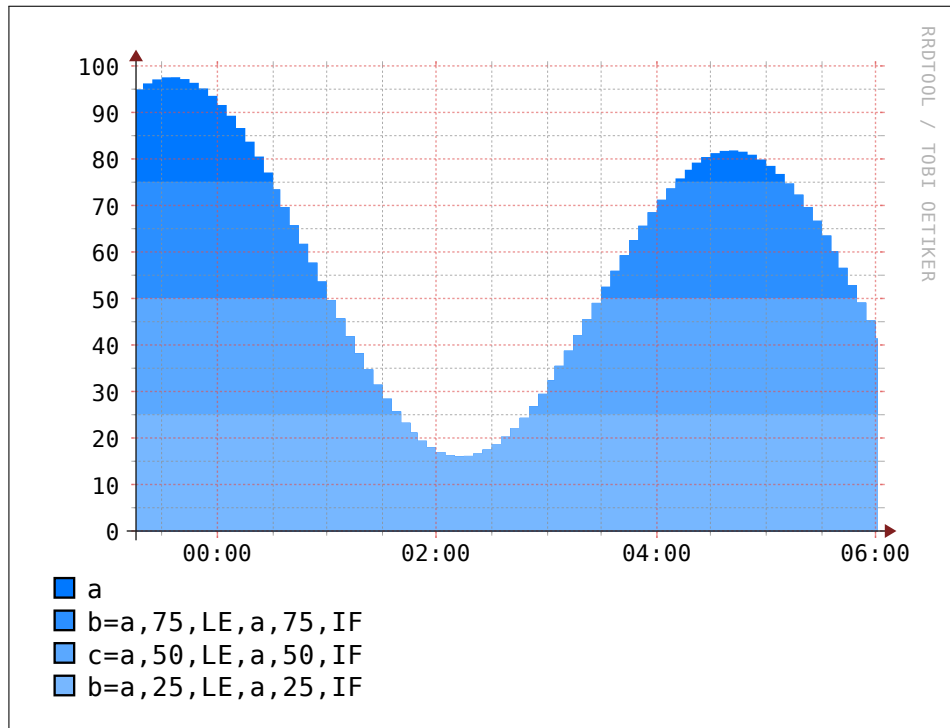
Another interesting option is to SHIFT the result of the TREND calculation back in time, so that it matches with the source data. This allows us to see more easily when there are 'outliers'

the IF function

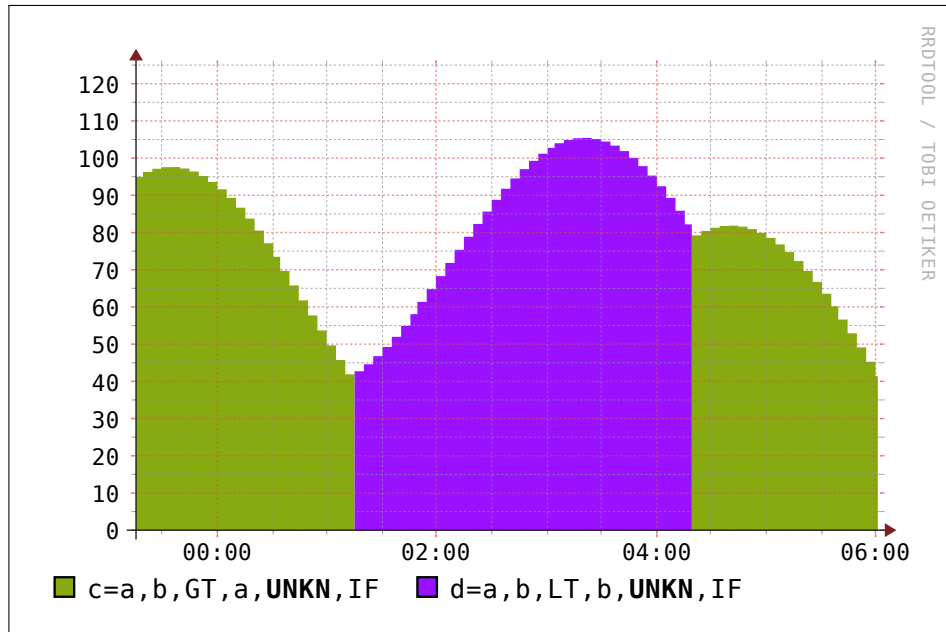


The IF function requires three items on the stack. It turns a, b, c, IF into if a then b else c . There is a bunch of operators that go along with the IF: LT less, LE - less or equal, EQ - equal, NE not equal, GE - greater or equal, GT - greater.

horizontal gradient

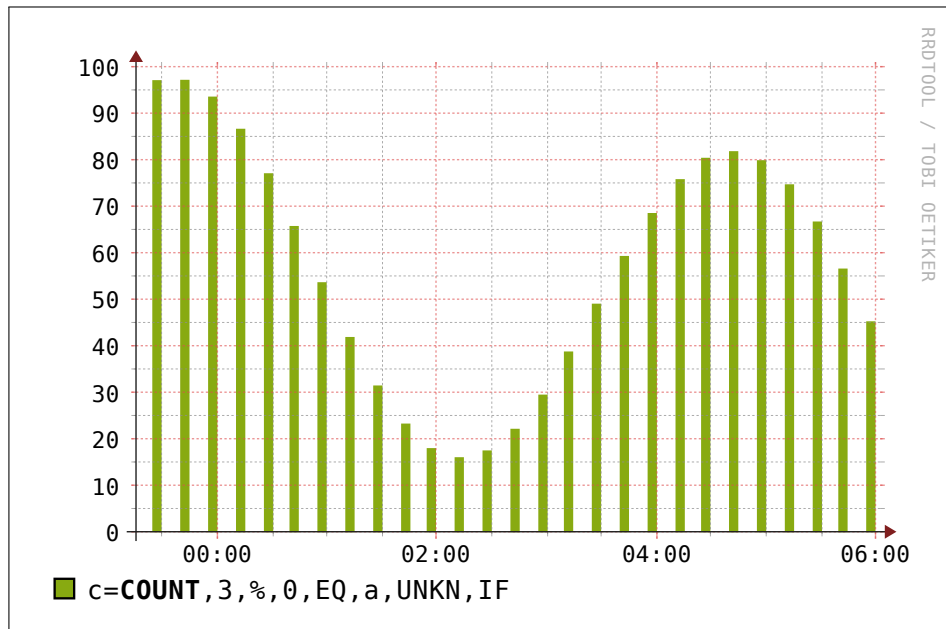


about invisibility



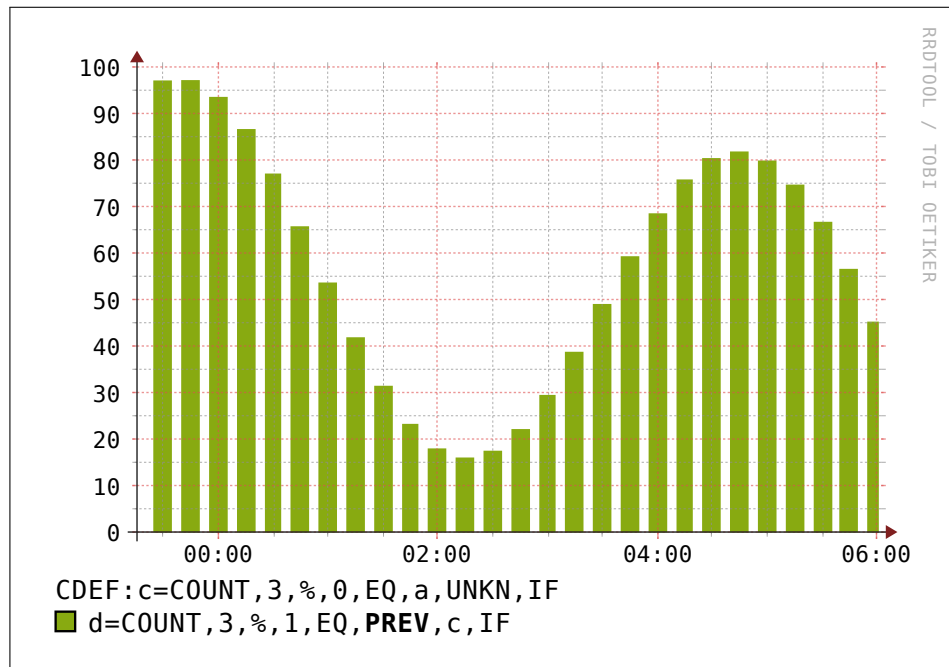
Unknown values are not drawn on the graph. Here we use trick to only show the largest values.

positional drawing count

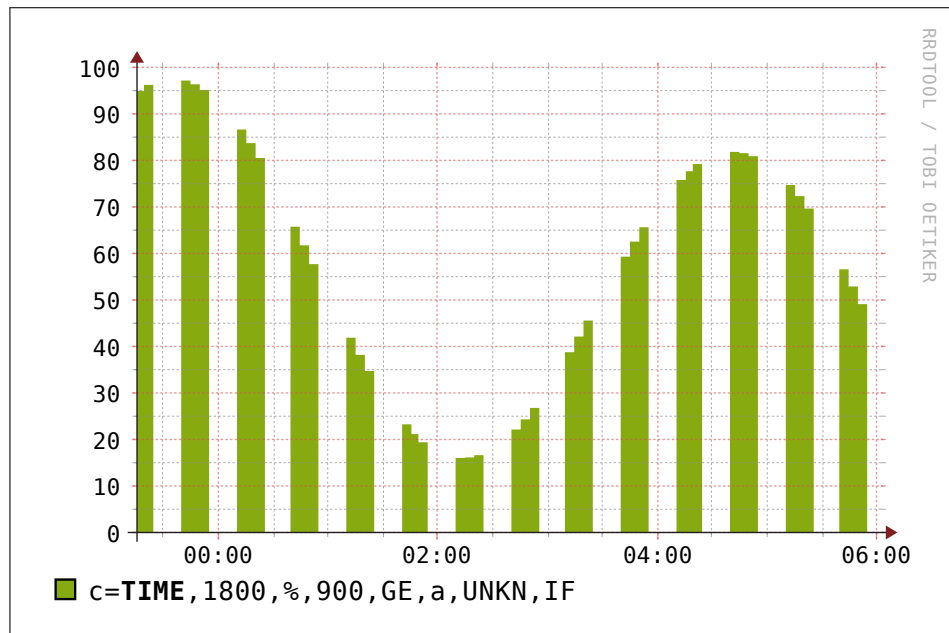


If you are into bar charts, you might fake them with this trick. COUNT, counts the values of the data set. We use this, together with the modulo operator to suppress the drawing of every third entry.

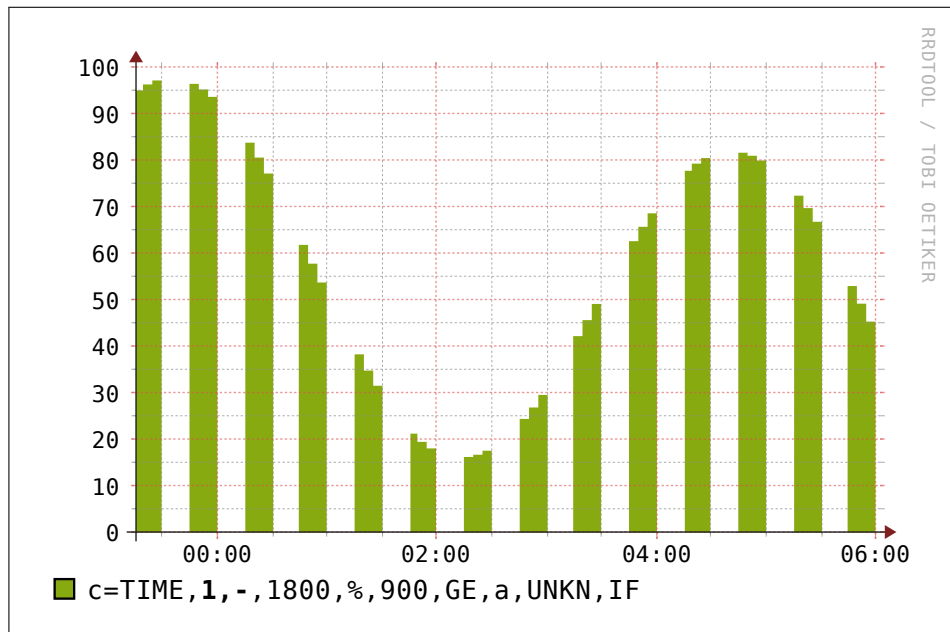
access the previous value



positional drawing time

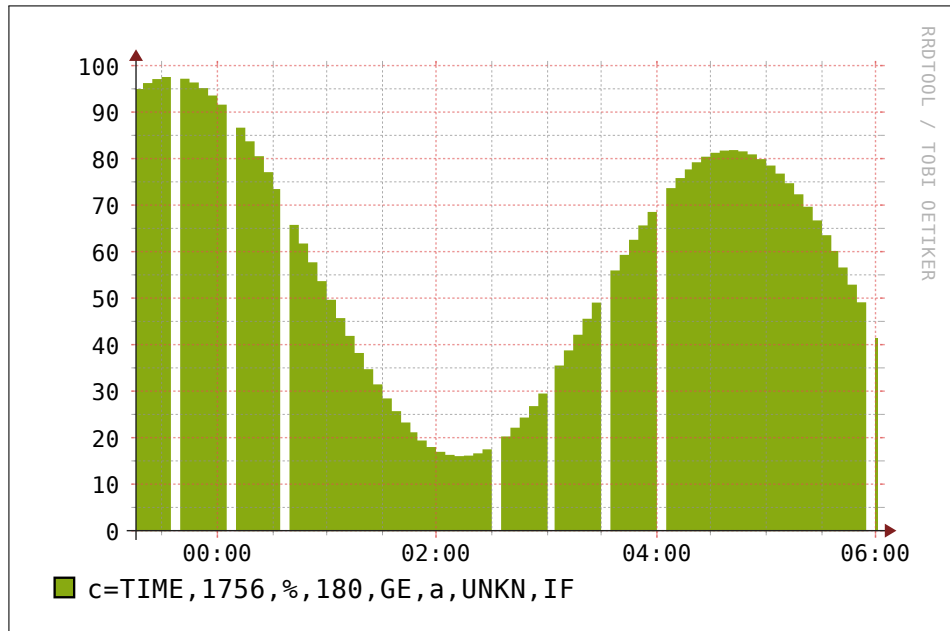


positional drawing time-shifting



There is also a function for accessing the Unix time (seconds since 1970) associated with the graph data. With it, you can make your stripes a fixed number of seconds wide.

time and resolution issues



Whenever RRDtool graph has to do math with data sets that come in different step sizes, it first to adjust the step sizes so that they match. To do this, it finds the greatest common divisor and uses it as the new step size.

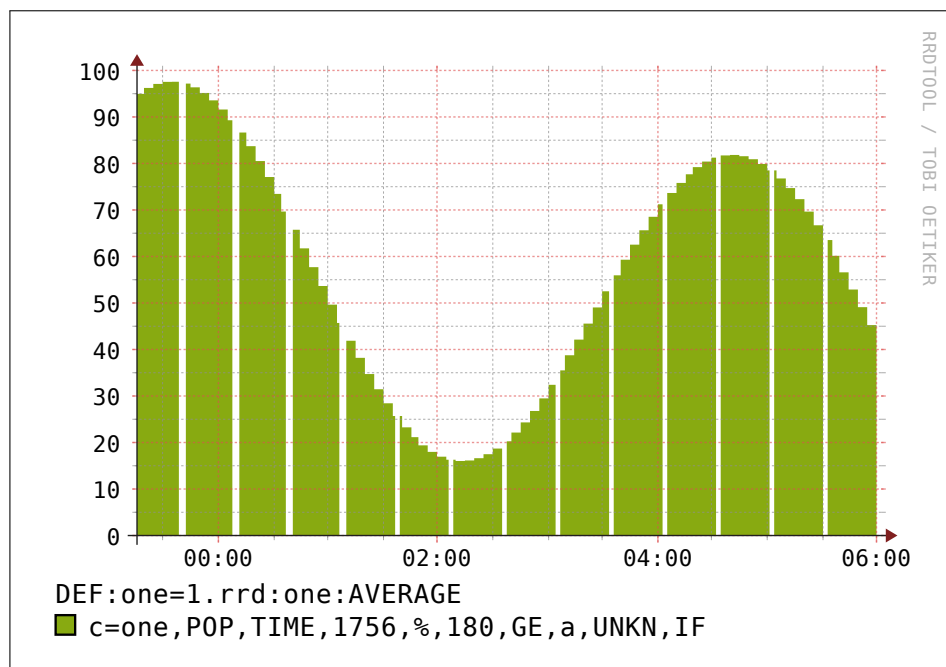
CDEF internals

- data may come in different resolutions
- all items in a CDEF must have the same resolution
- resolution is expanded to greatest common divisor (gcd)
- example: $\text{gcd}(6,9) = 3$, $\text{gcd}(1,6) = 1$

trick: an rrd with one a second step.

```
rrdtool create one.rrd --step=1
DS:one:GAUGE:2:U:U
RRA:AVERAGE:0.5:1:1
```

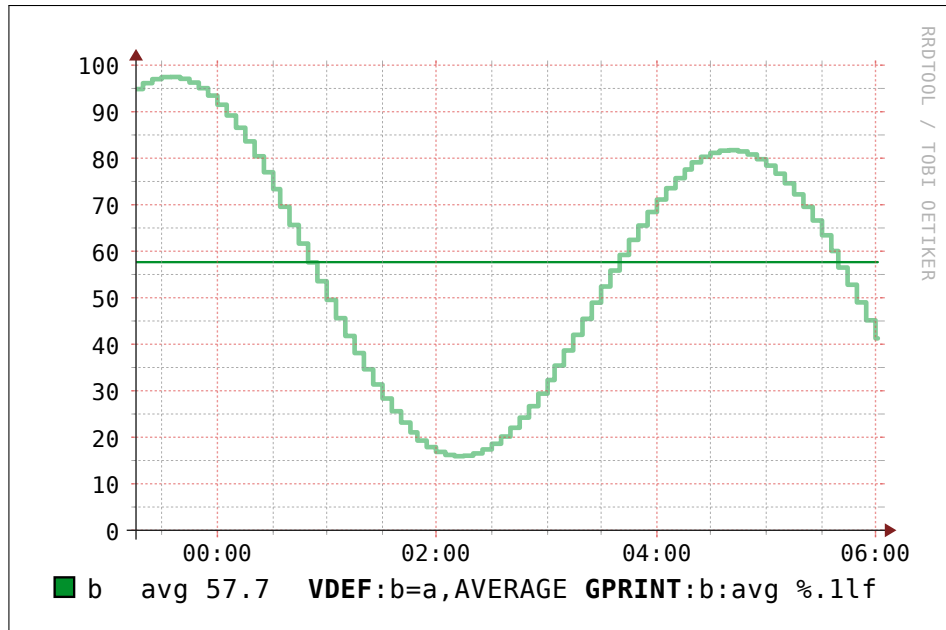
step=1 trick: high resolution cdef



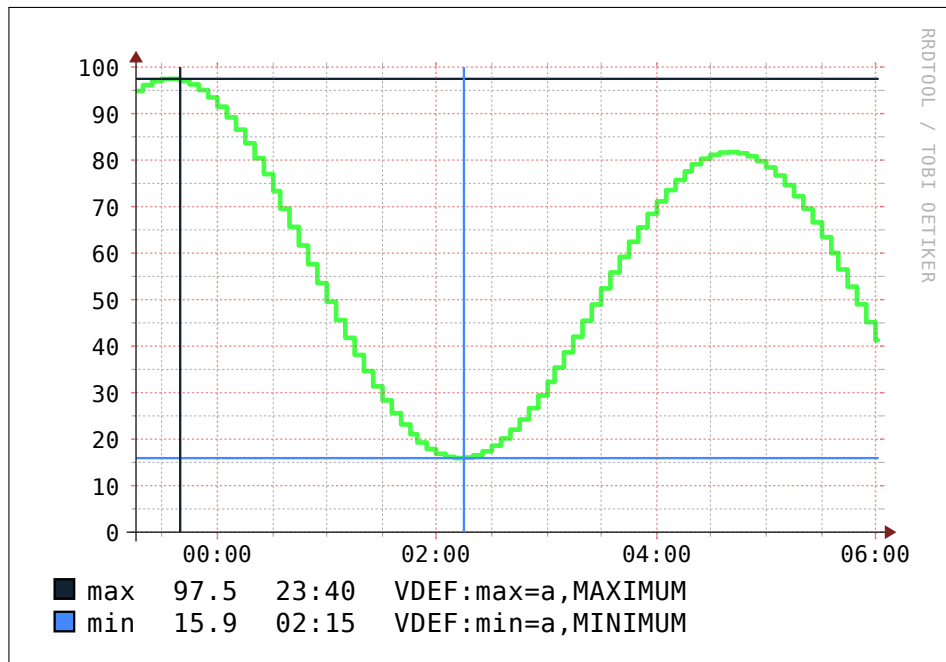
By introducing this special rrd with a “one second step” the greatest common divisor (gcd) becomes one.

4 Consolidation functions

finding the average



calculating min and max

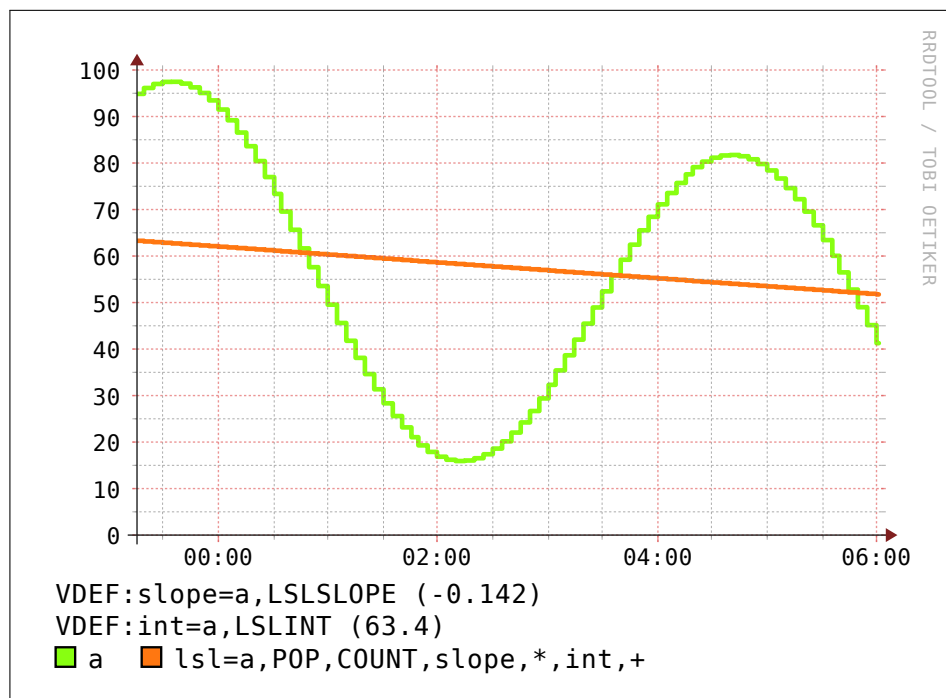


min max code example

```
LINE:a#456:a  
VDEF:max=a,MAXIMUM  
LINE:max#123  
VRULE:max#123:maximum  
GPRINT:max:%.1lf  
GPRINT:max:%H\:%M:strftime
```

A VDEF result has a value and a time assigned.

Least Squares Line ($y=x*m+b$)



5 Holt Winters Aberrant Behaviour Detection

about alert generation

- when something unexpected happens send an alert
- fixed thresholds are too wide a net
- moving averages weigh all data equal
- holt winters can predict the future
- and no one considers himself clever enough to use it

rrd - holt winters assumptions

- data is periodic in nature
- data has continuity
- data continuity is periodic
- recent data is more important

holt winters aberrant behavior

- prediction of future value and confidence band
- confidence band is like a standard deviation
- real value compared to predicted value +/- confidence band

With holt winters RRDtool will calculate a prediction and a confidence band (think of it as a standard deviation) for the current value. It will then compare the prediction with the actual value. If the actual value falls outside the confidence band of the predicted value (or some multiple of it), then a confidence band violation is registered. If multiple violations are registered within a configurable interval, RRDtool logs a failure.

- Keep it simple, go for HWPREDICT only when you start using holt winters in RRDtool.
- Every data set is different, tweaking is required.
- Know which knobs to turn.
- Use real data when experimenting.
- Use `rrdtool tune` and `rrdtool resize` to tweak settings.
- The FAILURES RRA is short!

holt winters parameters

RRA:HWPREDICT:rows:alpha:beta:period

alpha: adaption rate of the baseline (1 fast, 0 slow)

beta: adaption rate of the slope (1 fast, 0 slow)

period: how many steps in a period (use 1 to disable)

gamma: seasonal adaption rate of the baseline(alpha by default)

dev_gamma: seasonal adaption rate of the confidence band (gamma by default)

the gamma and confidence band are tunable with `rrdtool tune`

Reading [Brutlag's original paper](#) I wrote down the formulas he uses for calculating holt winters. This helped me quite a lot in understanding the relationships between alpha, beta, gamma and delta.

the rrdtool holt winters formula

a - baseline (RRA CDP Parameter)

b - slope (RRA CDP Parameter)

c - seasonal (SEASONAL RRA)

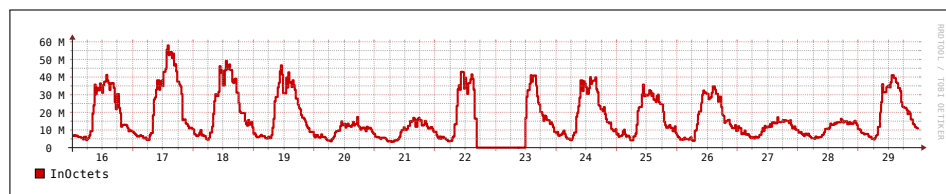
d - deviation (DEVSEASONAL RRA)

pred - predicted value

real - real value

$$\text{pred}\{\text{next}\} = \text{a}\{\text{now}\} + \text{b}\{\text{now}\} + \text{c}\{\text{next_prev_period}\}$$
$$\begin{aligned} \text{a}\{\text{now}\} = & \text{alpha} * (\text{real}\{\text{now}\} - \text{c}\{\text{now_prev_period}\}) \\ & + (1-\text{alpha}) * (\text{a}\{\text{prev}\} + \text{b}\{\text{prev}\}) \end{aligned}$$
$$\begin{aligned} \text{b}\{\text{now}\} = & \text{beta} * (\text{a}\{\text{now}\} - \text{a}\{\text{prev}\}) \\ & + (1-\text{beta}) * \text{b_prev} \end{aligned}$$
$$\begin{aligned} \text{c}\{\text{now}\} = & \text{gamma} * (\text{real}\{\text{now}\} - \text{a}\{\text{now}\}) \\ & + (1-\text{gamma}) * \text{c}\{\text{now_prev_period}\} \end{aligned}$$
$$\begin{aligned} \text{d}\{\text{now}\} = & \text{dev_gamma} * \text{abs}(\text{real}\{\text{now}\} - \text{pred}\{\text{now}\}) \\ & + (1-\text{dev_gamma}) * \text{d}\{\text{now_prev_period}\} \end{aligned}$$

hw demo: the test data



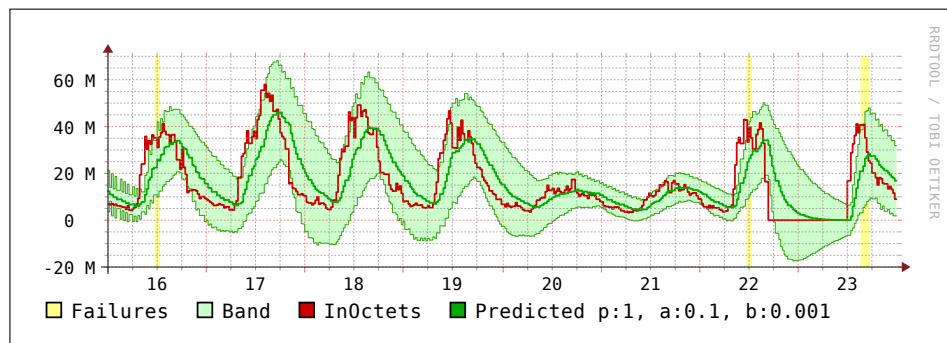
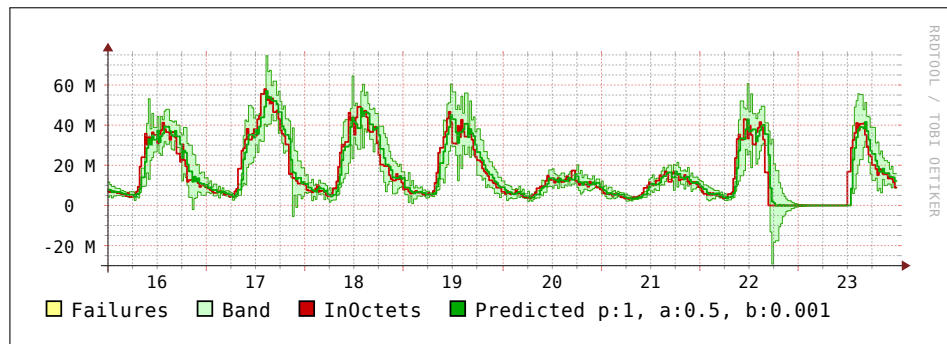
traffic at a peering point

drawing a hw graph

```
1 DEF:in=hw.rrd:in:AVERAGE
2 DEF:pred=hw.rrd:in:HWPREDICT
3 DEF:conf=hw.rrd:in:DEVPREDICT
4 DEF:fail=hw.rrd:in:FAILURES
5 TICK:fail#ff8:1:Failures
6 CDEF:lowconf=pred,conf,2,*,-,
7 LINE1:lowconf
8 CDEF:confwidth=conf,4,*
9 AREA:confwidth#cfc:Band:STACK
10 LINE0.1:0#3a1::STACK
11 LINE0.1:lowconf#3a1
12 LINE1:in#c00:InOctets
13 LINE1:pred#0a0:Prediction
```

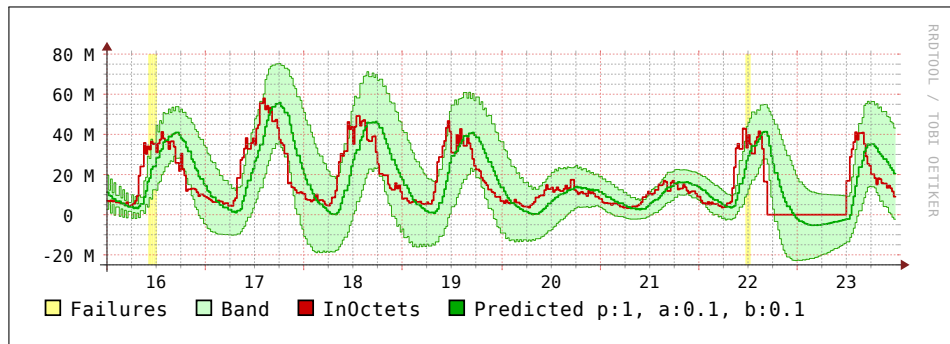
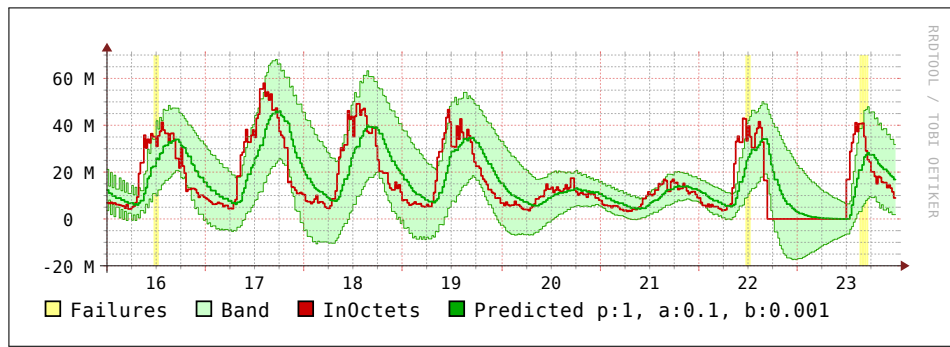
For starters we set the period to 1. This disables HW's ability to adjust to periodic behavior in the data but it lets us better observe the effects of the different parameter settings since the adjustment period is much shorter.

hw demo: alpha



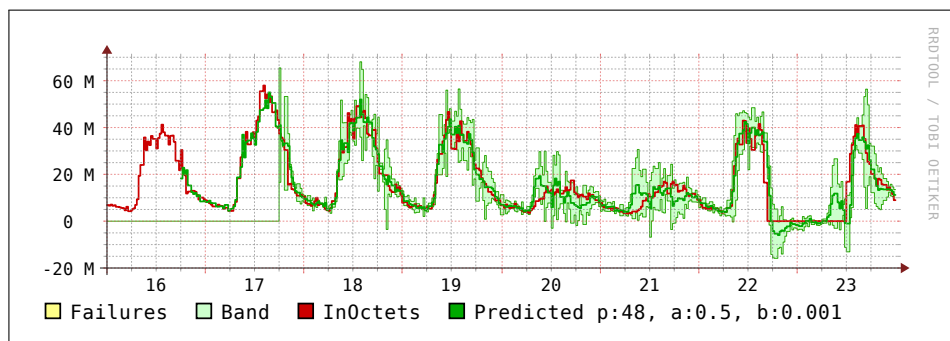
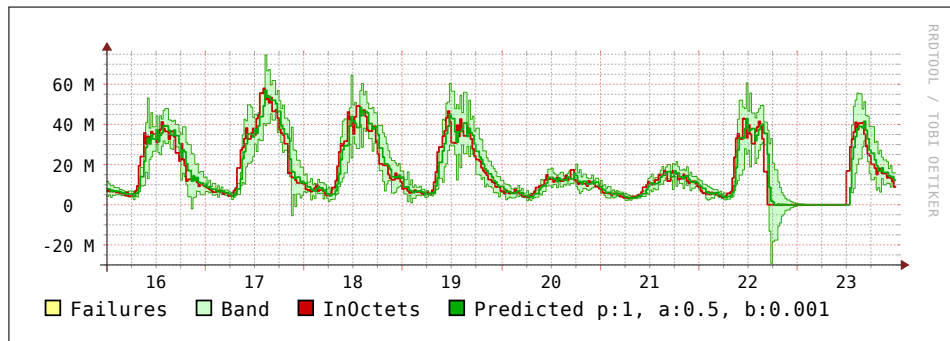
The smaller the alpha the slower the adaption. As the prediction is now generally off by quite a bit, this causes the confidence band to grow as well.

hw demo: beta

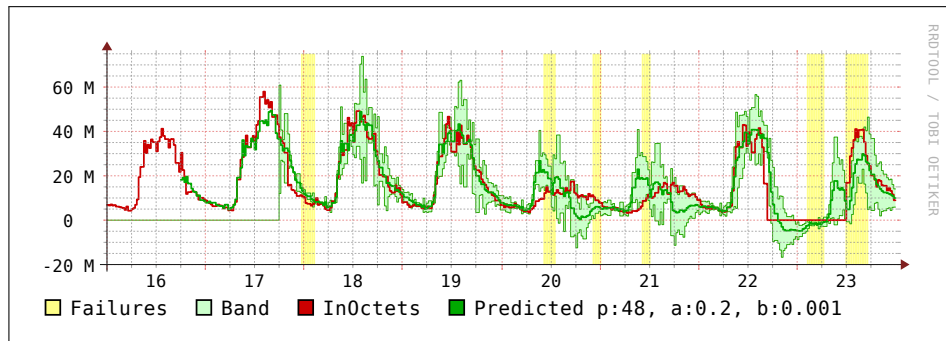
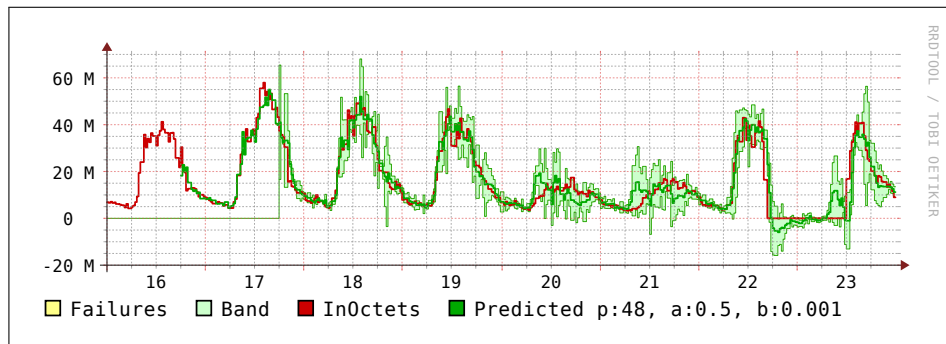


The larger the beta the 'heavier' the prediction becomes.

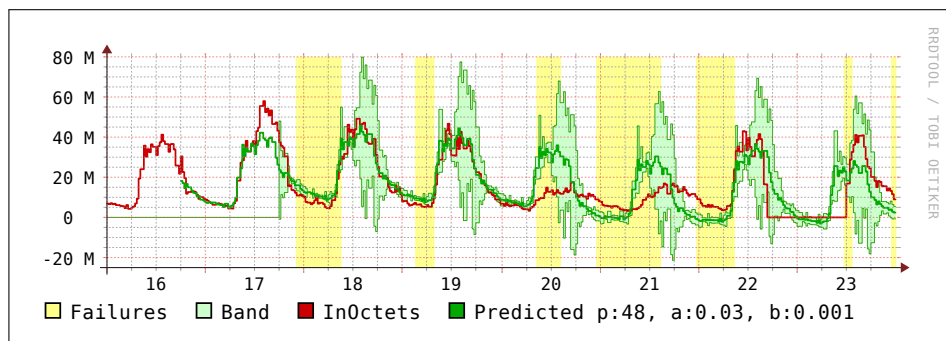
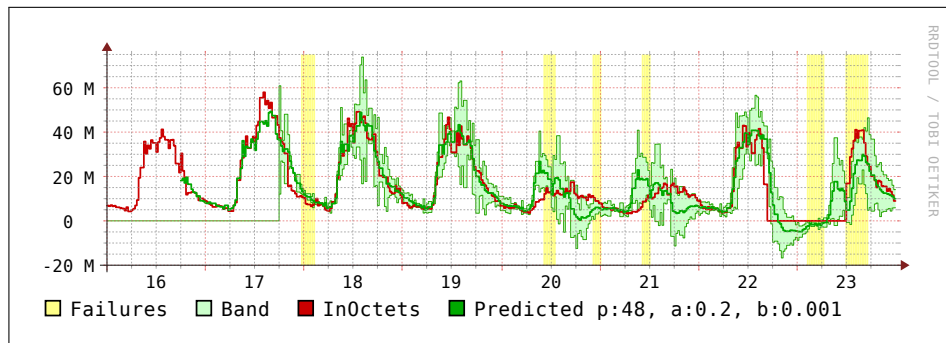
hw demo: period



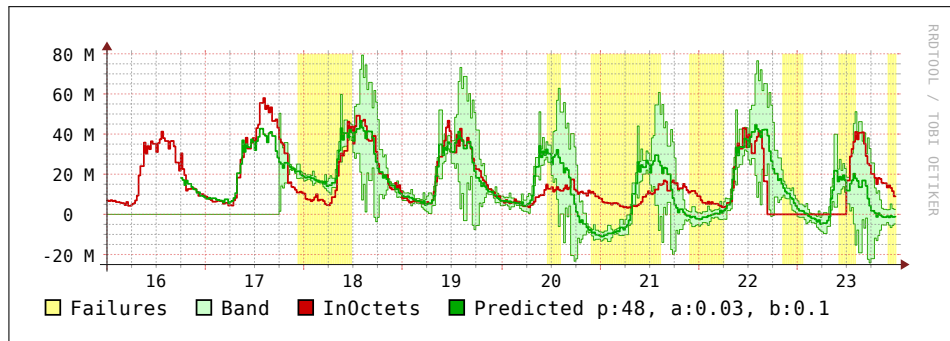
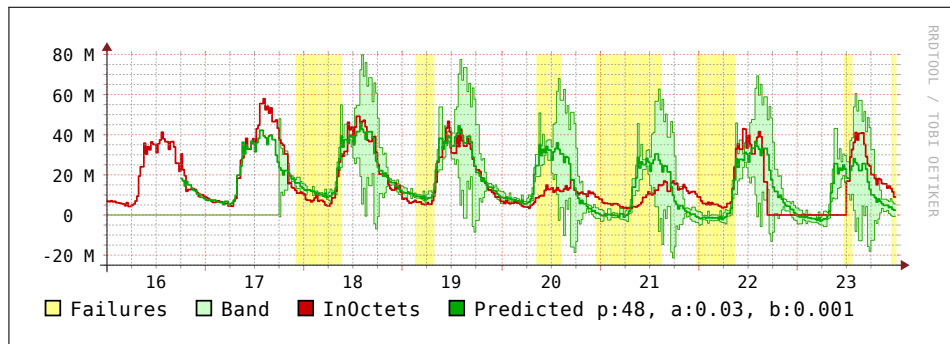
hw demo: tuning



hw demo: tuning II



hw demo: tuning III



6 The *v Interfaces

graphv script

```
1 #!/usr/bin/perl -w
2 use strict;
3 use lib qw( /scratch/rrd4/lib/perl );
4 use RRDs;
5 my $out = RRDs::graphv(
6     '-', '--start' => '00:00 20080916',
7     '--end' => 'start+8d',
8     '--lower-limit' => 0,
9     '--imgformat' => 'PDF',
10    'DEF:a=hw-demo.rrd:in:AVERAGE',
11    'LINE1:a#c00:InOctets');
12 my $ERROR = RRDs::error;
13 die "ERROR: $ERROR\n" if $ERROR;
14 map {
15     print $_.' = '.substr($out->{$_},0,8)."\n"
16 } sort keys %$out;
```

graphv output

```
1 graph_height = 100
2 graph_left = 51
3 graph_top = 22
4 graph_width = 400
5 image = %PDF-1.4
6 image_height = 163
7 image_width = 481
8 value_max = 60000000
9 value_min = 0
```

The image key only shows up when the graph file name is '-'.

v-interfaces

- rrdtool info
- rrdtool updatev
- rrdtool graphv

Using the output of updatev is much more efficient than re-reading the rrd file after running normal update against them.

7 RRD Caching Daemon

rrdcached — pushing rrd performance

- i/o comes in 4k chunks
- normal update is ~ 100 bytes
- grouping updates = performance for free
- data in memory
- journaling disaster recovery
- simple integration using environment variables
- communication via unix socket or ip
- limited security for remote operation
- no updatev support yet
- available with 1.4

created by Florian Forster and Kevin Brintnall

8 Future

Future RRDtool Features

- full remote support for rrdtool operations
- getopt to popt migration for thread-safety
- portable data format
- in-memory updates for cached to support updatev
- rrd internal journal for i/o optimization
- separation of database and charting features
- json interface and javascript charting frontend

Or anything else someone is willing to provide time or money for.

9 Examples

The size of an rrd - code

```
1 #!/usr/bin/perl
2 sub rrd_sizer {
3     my ($ds_cnt,$rra_sz,$rra_cnt) = @_;
4     system 'rrdtool', 'create', 'sizer.rrd',
5         map({ "DS:d${_}:GAUGE:600:U:U" } 1..$ds_cnt),
6         map({ "RRA:AVERAGE:0.5:1:$rra_sz" } 1..$rra_cnt);
7     my $size = -s 'sizer.rrd';
8     printf "DSs: %1d RRA Row: %1d RRAs: %1d == %3d byte\n",
9         $ds_cnt,$rra_sz,$rra_cnt,$size;
10    return $size;
11 }
12 #
13 #           DSs RRAs RRA Rows
14 my $base   = rrd_sizer 1, 1, 1;
15 my $ds     = rrd_sizer 2, 1, 1;
16 my $rra_sz = rrd_sizer 1, 2, 1;
17 my $rra_cnt = rrd_sizer 1, 1, 2;
18 printf "+1 DS:      %3d byte\n", ($ds - $base);
19 printf "+1 RRA Row:  %3d byte\n", ($rra_sz - $base);
20 printf "+1 RRA:      %3d byte\n", ($rra_cnt - $base);
```

the size of an rrd - result

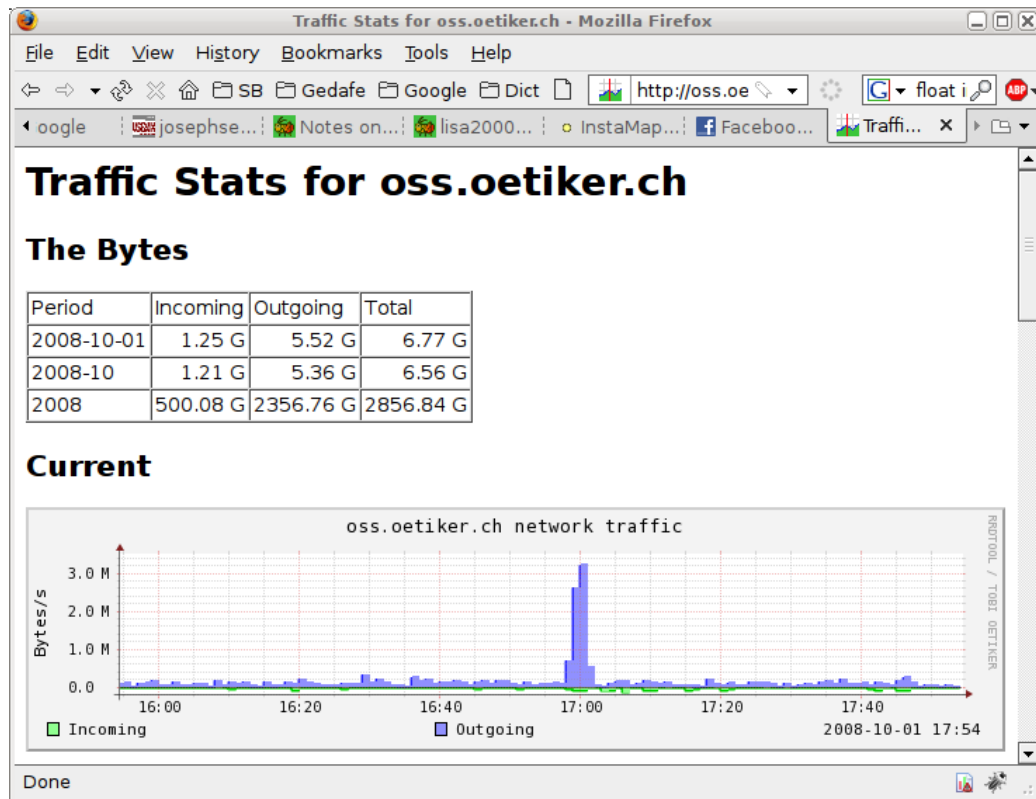
```
1 DSs: 1 RRA Row: 1 RRAs: 1 == 552 byte
2 DSs: 2 RRA Row: 1 RRAs: 1 == 872 byte
3 DSs: 1 RRA Row: 2 RRAs: 1 == 560 byte
4 DSs: 1 RRA Row: 1 RRAs: 2 == 752 byte
5 +1 DS:      320 byte
6 +1 RRA Row:   8 byte
7 +1 RRA:     200 byte
```

- overhead is minimal
- 8 byte per double
- ... per datasource
- ... per RRA
- ... per RRA row

The rrd format is highly efficient at storing non sparse data. The overhead for an extra RRA or DS is normally a few bytes on top of the 8 byte per double.

10 Real Live Example

The following example shows how to create a simple traffic grapher with a shell script for data acquisition, and an rrdcgi script to draw the graphs.



data acquisition

```
1 #!/bin/sh
2 # use from cron
3 # * * * * * /path/to/ifbyteget.sh eth0
4
5 PATH=/bin:/usr/bin
6 export PATH
7
8 cd /home/oepass/public_html/stats
9
10 if [ ! -f $1.rrd ]; then
11
12 rrdtool create $1.rrd \
13     --step=60 \
14     DS:in:DERIVE:70:0:100000000 \
15     DS:out:DERIVE:70:0:100000000 \
16     RRA:AVERAGE:0.5:1:1500 \
17     RRA:AVERAGE:0.5:60:10000 \
18     RRA:MIN:0.5:60:10000 \
19     RRA:MAX:0.5:60:10000 \
20     RRA:AVERAGE:0.5:1440:1000 \
21     RRA:MIN:0.5:1440:1000 \
```

```

22     RRA:MAX:0.5:1440:1000
23 fi
24
25 rrdtool update $1.rrd \
26     N:`grep $1: /proc/net/dev` \
27     | sed 's/.*://' | awk '{print $1:""$9}' `

```

This little bash script polls the network traffic counter from the linux proc tree and reformats it so that it can be fed to rrdtool.

rrdcgi: scripting for the poor

```

1  #!/usr/bin/env rrdcgi
2  <html>
3  <head>
4  <title>Traffic Stats for oss.oetiker.ch</title>
5  </head>
6  <body>
7  <h1>Traffic Stats for oss.oetiker.ch</h1>
8
9  <h2>The Bytes</h2>
10 <table border="1" cellspacing="0" cellpadding="2">
11 <tr><td>Period</td>
12     <td>Incoming</td>
13     <td>Outgoing</td>
14     <td>Total</td></tr>
15
16 <!--
17 <RRD::GRAPH -
18     --start="midnight"
19     --end="start+24h"
20     --imginfo=" "
21     DEF:in=lan.rrd:in:AVERAGE:step=1800
22     DEF:out=lan.rrd:out:AVERAGE:step=1800
23     VDEF:is=in,TOTAL
24     PRINT:is:"%0.2lf %s"
25     VDEF:os=out,TOTAL
26     PRINT:os:"%0.2lf %S"
27     CDEF:sum=in,out,+
28     VDEF:ss=sum,TOTAL
29     PRINT:ss:"%0.2lf %S"
30 >
31 -->
32
33 <tr><td><RRD::TIME::NOW %Y-%m-%d></td>
34     <td align="right"><RRD::PRINT 0></td>
35     <td align="right"><RRD::PRINT 1></td>
36     <td align="right"><RRD::PRINT 2></td></tr>
37
38 <!--
39 <RRD::GRAPH -
40     --start="<RRD::TIME::NOW %Y%m01>"
41     --end="now"
42     --imginfo=" "

```

```

43     DEF:in=lan.rrd:in:AVERAGE:step=1800
44     DEF:out=lan.rrd:out:AVERAGE:step=1800
45     VDEF:is=in,TOTAL
46     PRINT:is:"%0.2lf %s"
47     VDEF:os=out,TOTAL
48     PRINT:os:"%0.2lf %S"
49     CDEF:sum=in,out,+
50     VDEF:ss=sum,TOTAL
51     PRINT:ss:"%0.2lf %S"
52 >
53 -->
54
55 <tr><td><RRD::TIME::NOW %Y-%m></td>
56     <td align="right"><RRD::PRINT 0></td>
57     <td align="right"><RRD::PRINT 1></td>
58     <td align="right"><RRD::PRINT 2></td></tr>
59
60 <!--
61 <RRD::GRAPH -
62     --start="<RRD::TIME::NOW %Y0101>"
63     --end="now"
64     --imginfo=" "
65     DEF:in=lan.rrd:in:AVERAGE:step=1800
66     DEF:out=lan.rrd:out:AVERAGE:step=1800
67     VDEF:is=in,TOTAL
68     PRINT:is:"%0.2lf %s"
69     VDEF:os=out,TOTAL
70     PRINT:os:"%0.2lf %S"
71     CDEF:sum=in,out,+
72     VDEF:ss=sum,TOTAL
73     PRINT:ss:"%0.2lf %S"
74 >
75 -->
76
77 <tr><td><RRD::TIME::NOW %Y></td>
78     <td align="right"><RRD::PRINT 0></td>
79     <td align="right"><RRD::PRINT 1></td>
80     <td align="right"><RRD::PRINT 2></td></tr>
81 </table>
82
83 <h2>Current</h2>
84
85 <RRD::SETVAR start -2h>
86 <RRD::SETVAR end now>
87 <RRD::INCLUDE graph.inc>
88
89 <h2>Day</h2>
90
91 <RRD::SETVAR start -24h>
92 <RRD::SETVAR end now>
93 <RRD::INCLUDE graph.inc>
94
95 <h2>7 Days</h2>
96
97 <RRD::SETVAR start -7d>

```



```

98 <RRD::SETVAR end now>
99 <RRD::INCLUDE graph.inc>
100
101 <h2>Month</h2>
102
103 <RRD::SETVAR start -30d>
104 <RRD::SETVAR end now>
105 <RRD::INCLUDE graph.inc>
106
107 <h2>This Year</h2>
108
109 <RRD::SETVAR start "Jan1">
110 <RRD::SETVAR end "Dec31">
111 <RRD::INCLUDE graph.inc>
112
113 <h2>Last Year</h2>
114
115 <RRD::SETVAR start "Jan1-365d">
116 <RRD::SETVAR end "Dec31-365d">
117 <RRD::INCLUDE graph.inc>
118
119 </body>
120 </html>

```

rrdcgi: include file function

```

1 <p>
2 <RRD::GRAPH lan<RRD::GETVAR start>.png
3     --title="oss.oetiker.ch network traffic"
4     --vertical-label=Bytes/s
5     --start="<RRD::GETVAR start>"
6     --end="<RRD::GETVAR end>"
7     --width=600
8     --height=100
9     DEF:in=lan.rrd:in:AVERAGE
10    CDEF:nin=in,-1,*
11    LINE1.5:nin#00d000
12    AREA:nin#90ff90:Incoming
13    DEF:out=lan.rrd:out:AVERAGE
14    LINE1.5:out#2020ff
15    AREA:out#9090ff:Outgoing
16    LINE0.5:0#000
17    COMMENT:"<RRD::TIME::NOW '%Y-%m-%d %H:%M'>\j"
18 >
19 </p>

```

RRDtool's rrdcgi is a very simple scripting engine, that can pick up pseudo xml elements from an html file and execute the corresponding rrdtool commands. In this example we use environment variables and an include file to save us from typing in the same graph definition over and over again.

Tobias Oetiker <toebi@oetiker.ch>