Getting Started with RRDtool

Cuddletech TekRef Series

Ben Rockwood, Cuddletech <benr@cuddletech.com>

Revision History		
Revision v1.0	April 27th 2004 Initial Document	benr
Revision v1.1	May 10th 2004 Final sections added	benr

RRDtool is a powerful replacement for MRTG, but it's complicated and extremely confusing the first time you use it. This paper hopes to make your first experience a little less mind numbing.

Table of Contents

Introduction

RRDtool was written by Tobi Oetiker, the author of MRTG. It is, effectively, the next generation of MRTG, with a complete reimplementation of MRTGs graphing and logging features. MRTG works great for simple network monitoring, but thats really all it was originally intended to do. These days people are using MRTG to monitor everything from router bandwidth, to memory and disk statistics, to the number of times the dog goes in and out of the house.

There are a number of drawbacks to MRTG however. On the graphing front, MRTG graphs always use a Y axis starting at 0, if you only want to see the relevant values in a range (for temps you might only want to graph from 50F to 90F) your out of luck. You are significantly limited to the number of different values that can be graphed, if you want to see the network throughput of 10 different servers your probably going to be

forced to use 10 different graphs. The list goes on and on. The point is that RRDtool fills in the gaps that MRTG leaves wanting, and provides for open customization that was difficult if not impossible before.

But, MRTG has one thing that RRDtool doesn't... *simplicity*. Many Network Admins who know more about air purifiers than UNIX systems are using MRTG on a regular basis, thanks to tools like cfgmaker and indexmaker, coupled with a simple and basically straightforward config syntax. RRDtool isn't quite so simple though, at least at first. Almost all functions are provided by a single tool: **rrdtool**. The same program is used to create databases, modify, tune and update them, generate graphs, and even make backup dumps.

Lets look at the basic flow you follow when using RRDtool:

Overview of steps to use RRDtool for data graphing

1. Create an empty RRD database using **rrdtool create**.
2. Utilize a script and/or the cron to add data to the database using **rrdtool update**.
3. Generate, usually via script, custom output graphs using **rrdtool graph**.

In this tutorial we'll look quickly at each step, so that you can get off and generating your own output.

Creating an initial RRD

RRD stands for Round Robin Database, and it is what it sounds like. There are a fixed number of records in the database and once the last record has been written in the database the next update goes to the first record, and around and around it goes. In this way, your databases will never grow out of control. The only downside to this is that, obviously, you've got to know how much data you'll want to look at historically ahead of time so that when you generate your database you have enough data. You may want a days worth of info, or even months.

When creating an RRD database we'll need to specify a couple things, namely one or more Data Sources and one or more Round Robin Archives. The data source (DS) defines what type of data is accepted and some boundaries on what constitutes good data. The round robin archives (RRA) can almost be thought of as views, it defines the different ways we can store and retrieve data.

The following is an example of database creation. In this example I am setting up an RRD to monitor TempTrax temperatures.

```
[benr@nexus TempTrax-RRD]$ rrdtool create temptrax.rrd \
--start N --step 300 \
DS:probe1-temp:GAUGE:600:55:95 \
DS:probe2-temp:GAUGE:600:55:95 \
DS:probe3-temp:GAUGE:600:55:95 \
DS:probe4-temp:GAUGE:600:55:95 \
RRA:MIN:0.5:12:1440 \
RRA:MAX:0.5:12:1440 \
RRA:AVERAGE:0.5:1:1440
```

Figure 1. TempTrax RRD Creation Command

The tool **rrdtool** is called with the argument **create** followed by the RRD filename of the database to be created. The next two arguments specify the time at which the database starts, measured in seconds since the Epoch, and the step time measured in seconds which is the interval between database updates. In this case I'm using "N" as the start time which tells rrdtool to use "now" as the time, and my step interval is 300 seconds (5 minutes).

The "DS" lines specify our different data sources. In this case I want to monitor 4 different temperature probes which I've named sequentially. The following field ("GUAGE") species the Data Source Type (DST) which is one of GUAGE, COUNTER, DERIVE or ABSOLUTE. The Guage DST works like you'd expect and is generally the best choice. Counters continuously increase, Derives store a derivative of the last and the current value, and Absolutes store values which reset after each reading. The last 3 fields of the Data Source lines specify the minimal heartbeat and both min and max values. The minimal heartbeat is a value measured in seconds after which the values is said to be unknown (think of it as a timeout). The min and max specify a range for "good" values; if a value is outside of this range it is said to be unknown.

The "RRA" lines specify different round robin archives. These are likes views, by which the data are stored. Inside the RRD database file each RRA is stored separately, with a predefined number of records. Each time we "update" our database we are adding a Primary Data Point (PDP), which are then combined together and put into our RRA based on a Consolidation Function (CF) that determines what the actual value thats written is. In the above example the first field specifies that we're defining an RRA. The second field specifies the CF that is used, one of AVERAGE, MIN, MAX or LAST. The third field specifies the XFiles Factor (XFF), this is the percentage of PDPs that can be unknown without making the recorded value unknown. The fourth field is the number of PDPs that will make up the recorded value. The final field specifies the number of records this RRA has.

Looking again at the example above, I'm creating a new RRD named `temptrax.rrd` that starts "Now" and is updated (stepped) every 300 seconds (5 minutes). The RRD contains

4 different Data Sources, 1 per probe of the type GAUGE. If the Data Source isn't updated every 600 seconds (10 minutes) or if the value is not between 55 and 95, then the value is considered to be in error and is written as unknown. Then, 3 different RRA's are specified. Two of the RRA's record the MIN and MAX values using 12 PDP's allowing 50% of them to be unknown. We're storing 1440 records in these RRAs which means that because we're updating every 5 minutes (step) and we're using 12 PDPs (each update is a PDP), which means we're adding a record every hour (5 mins * 12) and we're storing 1440 records meaning that we will have 60 days (1440hrs/24hrs) worth of data in our RRA. For each of these the minimum value and the maximum value from the collected PDPs will be used as the recorded value. In the last RRA defined we're using the average of our collected PDPs, still allowing for 50% unknowns, but we're using only 1 PDP per record, which means we're storing every update. We're allowing for 1440 records which means this RRA is storing (1440/12updatesperhouse/24hrs) 5 days worth of data. In this later case because we're using a single PDP the CF isn't really important and we probably should have used LAST just for cleanliness.

Here's another breakdown of the DS and RRA arguments:

Data Source Fields: DS:DS-Name:DST:HeartBeat:Min:Max

DS	Defines a Data Source Field.
DS-Name	The name of this Data Source.
DST	Defines the Data Source Type. Can be GAUGE, COUNTER, DERIVE or ABSOLUTE.
HeartBeat	Defines the minimum heartbeat, the maximum number of seconds that can go by before a DS value is considered unknown.
Min	The minimum acceptable value. Values less than this number are considered unknown. *This is optional. Specify "U" (unknown) to not set a min*
Max	The maximum acceptable value. Values exceeding this number are considered unknown. *This is optional. Specify "U" (unknown) to not set a max*

Round Robin Archives: RRA:CF:XFF:Steps:Rows

RRA	Defines a Round Robin Archive.

CF Consolidation Function. Can be AVERAGE, MIN, MAX, or LAST.

XFF Defines XFiles Factor, the number of data points that can be anally probed by martians before RRD gives a crap.

Steps Defines how many Primary Data Points (PDPs) are consolidated using the Consolidation Function (CF) to create the stored value.

Rows Defines the number of Rows (records) stored in this RRA.

Adding Timed Data

Adding data points (**rrdtool update**) can be the easiest or hardest part of configuring RRD. This is where you actually insert the data you want to plot into the RRD. The overall idea, however, is extremely simple. Look at the form:

```
rrdtool update <file.rrd> timestamp:val1[:val2:...]
```

Typically this command is placed at the end of a script that nabs all your values via SNMPget or whatever method you are using. The form above is simple, **rrdtool update** (it should be noted you can also just use rrdupdate) followed by the RRD filename and an update string that starts with the timestamp of these values (as with create, you can use "N" to mean "Now"), and is followed by a list of colon separated values. You should be adding 1 value per Data Source, as defined in your RRD. The also must be added in order. You can actually use the **rrdtool update --template ds2:ds1:ds3** command form to change the default order of the data sources, but otherwise they are added in the order they were created in your RRD.

Here is an example update script which I run from cron every 5 minutes. It uses a Nagios plugin to get temps from a TempTrax E, does some string manipulation to cut off the output I don't want, and then updates RRD with the values.

```
#!/usr/local/bin/perl
# RRD Update Script: update_rrd_temps.pl

$HOST = "10.10.0.90";
$PATH = "/home/benr/RRD/TempTrax-RRD";
$NumProbes = 4;

for($i=1; $i <= $NumProbes; $i++) {
  $x = `${PATH}/check_temptraxe -H ${HOST} -p ${i}`;
  $x =~ s/^.*([0-9.]{4}).*$/$1/;
  chomp($x);
  push(@TEMPS,$x);
}
```

```
`/usr/local/rrdtool-1.0.48/bin/rrdtool update ${PATH}/temptrax.rrd
 "N:$TEMPS[0]:$TEMPS[1]:$TEMPS[2]:$TEMPS[3]"`;
```

Figure 2. TempTrax RRD Update Script

Obviously, the more complex you method of getting values is the more complex your update script will be. Typically this script is type of script is run from cron, however you could just use sleeps and a loop to keep the monitoring script running all the time. This is one of the nice things about RRDtools flexibility, you can get much more creative (for good or bad, you decide).

Generating Graphs

Generating graphs is a learned ability. It takes some time and experimentation, but isn't too tricky once you've done it once or twice. Graphs are created using **rrdtool graph**. There are tons of options and ways to configure graph output, but we'll just look at the most commonly used ones. Here is an example to go with the temp monitoring stuff we did earlier:

```
rrdtool graph mygraph.png -a PNG --title="TempTrax" \
--vertical-label "Deg F" \
'DEF:probe1=temptrax.rrd:probe1-temp:AVERAGE' \
'DEF:probe2=temptrax.rrd:probe2-temp:AVERAGE' \
'DEF:probe3=temptrax.rrd:probe3-temp:AVERAGE' \
'DEF:probe4=temptrax.rrd:probe4-temp:AVERAGE' \
'LINE1:probe1#ff0000:Switch Probe' \
'LINE1:probe2#0400ff:Server Probe' \
'AREA:probe3#cccccc:HVAC' \
'LINE1:probe4#35b73d:QA Lab Probe' \
'GPRINT:probe1:LAST:Switch Side Last Temp\: %2.1lf F' \
'GPRINT:probe3:LAST:HVAC Output Last Temp\: %2.1lf F\j' \
'GPRINT:probe2:LAST:Server Side Last Temp\: %2.1lf F' \
'GPRINT:probe4:LAST:QA Lab Last Temp\: %2.1lf F\j'
```

Figure 3. TempTrax Graph Creation Command

This looks worse than it is. The DEF lines define a Data Source (DS) to use. The LINE and AREA lines define the actual plotting methods. And the GPRINT lines define text that I want rendered onto the graph.

The default image output format is GIF, so using the -a allows you to choose between GIF, PNG and GD formats. --title is the string that is printed onto the top of your graph. -

-verital-label is an optional string that is printed on the Y axis, typically to denote the metric.

The DEF line DEFines a virtual name (vname) for an RRD Data Source (DS). It might seem odd that you need to define a virtualname, instead of just using the original DS name, but because you define the RRD filename here, you can define multiple DS's from different RRDs, which could be a problem if multiple RRDs have DS's by the same name; this method gets us around that. So the 2nd argument in the DEF line is vname=RRDfilename, followed by the actual original DS name, followed by the Consolidation Function (CF) type specified in your RRA. In this way, we are telling the graphing tool exactly what we want, from what file, from which RRA/view.

The next set of lines are LINE1 and AREA. These are the actual lines on the graph. Lines come in 3 different varieties, LINE1, LINE2, and LINE3. These are actually just Small, Medium, Large. LINE1 draws a 1px line, LINE2 draws a thicker line, and Line3 the thickest, otherwise they all work the same way. AREA is used to define a line that is "filled in". The big green area from 0 to your value in MRTG are AREAs. The form for all these types is the type, followed by the vname and a color specified in hex RGB value. Do note that if you do not specify a color the line won't show up! The final argument is the name of the line itself, this is printed at the bottom of the graph with the color key.

Finally, the GPRINT line allow us to write other stuff on the outputted graph. Generally it's nice to see the current values printed, so we'll draw those on. The first argument obviously is GPRINT, the second is the vname. The third argument is the CF type for data printed, here I'm using LAST because I want the most current value. And the final argument is the line thats actually printed. You will need to escape out colons here if you them so they don't chop off your line. Also, \j can be used to justify a line (push it to the right side of the graph). Inserting numeric values are done like in many programming languages: %1.1lf would give you a single precision floating point number, like 5.2 or 0.6, and %2.4lf would be a number like 23.1412. Whole number can be rounded with %2.0lf.

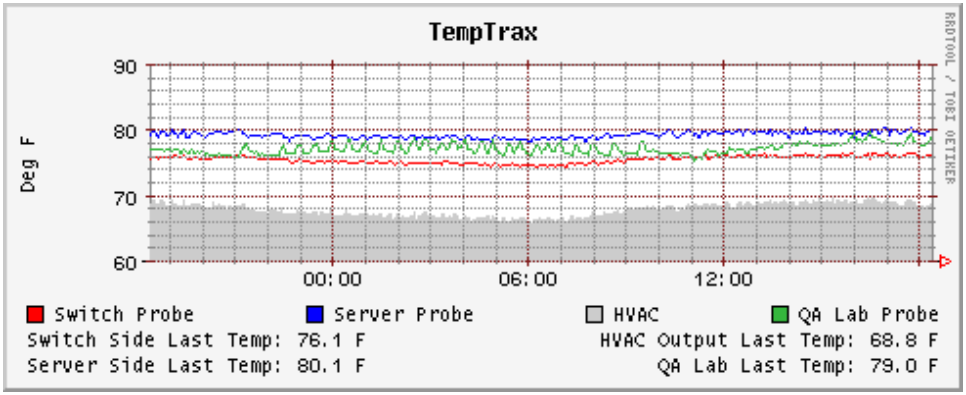

Figure 4. Generated TempTrax Graph

I used two separate scripts (PERL for the update, and Borne for the graphing) but ultimately it is probably cleaner to combine the two so that you are only running one script from cron.

Another look, start to finish

If your the type of reader who skip most of the text in an article and only reads the figures and examples, then this is the section to actually read.

The Network Admin at my company currently is using MRTG to monitor 4 interconnecting T1's we use to provide cooperate connectivity. Currently there are 6 different MRTG graphs (4 links, plus an aggregate and the faster ether interface on the router). I'd like to give him all this information on a single graph.

I want to grab the IF-MIB::ifInOctets and IF-MIB::ifOutOctets OIDs from the router, and I need to do it for the 4 T1 interfaces, the 1 FastEther and the 1 virtual multilink. So when I set up my RRD I'm going to need Data Sources for each in and out value (so thats 12 total). As for RRA, I want to hold the average per hour for 6 months and the last value every step interval going back a week. This way I can provide a graph for the last month or more and a current graph. That means we want 1 PDP for the LAST with 4032 records to hold 2 weeks (12 steps/hour * 24hrs * 14days) and 4320 records to hold our 6 months worth of hourly averages (24hrs * 180days).

I'll setup the RRD in the following way:

```
rrdtool create wan-thruput.rrd \
--start N --step 300 \
DS:Serial00_In:COUNTER:600:U:U \
DS:Serial01_In:COUNTER:600:U:U \
DS:Serial10_In:COUNTER:600:U:U \
DS:Serial11_In:COUNTER:600:U:U \
DS:FastEth20_In:COUNTER:600:U:U \
DS:MutliLink_In:COUNTER:600:U:U \
DS:Serial00_Out:COUNTER:600:U:U \
DS:Serial01_Out:COUNTER:600:U:U \
DS:Serial10_Out:COUNTER:600:U:U \
DS:Serial11_Out:COUNTER:600:U:U \
DS:FastEth20_Out:COUNTER:600:U:U \
DS:MutliLink_Out:COUNTER:600:U:U \
RRA:LAST:0.5:1:4032 \
RRA:AVERAGE:0.5:12:4320
```

Figure 5. Bandwidth RRD Creation Command

Now, time to start dropping data into it. I hacked around a bit and built the following script:

```
#!/usr/local/bin/perl
### WAN RRD Script

$RRDTOOL = "rrdtool";
$TARGET_RRD = "wan-thruput.rrd";
$SNMPGET = "snmpget -v2c -Ovq -c public 10.1.2.3";

for($i=1; $i <= 11; $i++){
        my $in = `${SNMPGET} IF-MIB::ifInOctets.${i}`;
        my $out = `${SNMPGET} IF-MIB::ifOutOctets.${i}`;

        chomp($in) && chomp($out);

        push(@T1_IN,$in);
        push(@T1_OUT,$out);
}

`${RRDTOOL} update $TARGET_RRD N:${T1_IN[0]}:${T1_IN[1]}:${T1_IN[2]}
:${T1_IN[3]}:${T1_IN[6]}:${T1_IN[10]}:${T1_OUT[0]}:${T1_OUT[1]}
:${T1_OUT[2]}:${T1_OUT[3]}:${T1_OUT[6]}:${T1_OUT[10]}`;
```

Figure 6. Bandwidth RRD Update Script

I've taken the full pathnames out of the above script to make it fit on printed pages, but this script will grab all the input and output octet counters (an octet is a group of 8 bits) and put them in our RRD.

Now to graph it. First we'll create the current graph using the latest numbers. We'll invoke RRD using the **rrdtool graph** tool. I'm using a shell script that can be executed from cron, that looks like this:

```
#!/bin/bash

rrdtool graph wanoutput.png -a PNG \
--title="HS T1 Links" --vertical-label "Bytes" \
--height 150 \
'DEF:s00in=wan-thruput.rrd:Serial00_In:LAST' \
'DEF:s01in=wan-thruput.rrd:Serial01_In:LAST' \
'DEF:s10in=wan-thruput.rrd:Serial10_In:LAST' \
'DEF:s11in=wan-thruput.rrd:Serial11_In:LAST' \
'DEF:fe20in=wan-thruput.rrd:FastEth20_In:LAST' \
'DEF:mlin=wan-thruput.rrd:MutliLink_In:LAST' \
'DEF:s00out=wan-thruput.rrd:Serial00_Out:LAST' \
'DEF:s01out=wan-thruput.rrd:Serial01_Out:LAST' \
'DEF:s10out=wan-thruput.rrd:Serial10_Out:LAST' \
'DEF:s11out=wan-thruput.rrd:Serial11_Out:LAST' \
'DEF:fe20out=wan-thruput.rrd:FastEth20_Out:LAST' \
```

```
'DEF:mlout=wan-thruput.rrd:MutliLink_Out:LAST' \
'HRULE:193000#0000ff' \
'HRULE:386000#0000ff' \
'HRULE:579000#0000ff' \
'HRULE:772000#0000ff' \
'AREA:mlin#66FF99:MultiLink In' \
'AREA:mlout#FFFF33:MultiLink Out' \
'LINE1:fe20in#000000:FastEther In' \
'LINE1:fe20out#ff0000:FastEther Out' \
'GPRINT:s00in:LAST:XO T1 A In\: %6.0lf bytes ' \
'GPRINT:s01in:LAST:XO T1 B In\: %6.0lf bytes\j' \
'GPRINT:s10in:LAST:SBC T1 A In\: %6.0lf bytes' \
'GPRINT:s11in:LAST:SBC T1 B In\: %6.0lf bytes\j'
```

Figure 7. Bandwidth Graph Creation Script

Again, for formating reasons I've removed full paths from the above script, I highly suggest you use fully qualified paths for all files in your script.

So, in the above script we're using PNG as the output format for the graph, and naming it "wanoutput.png". The graph is labeled "HS T1 Links" and the output image is 150px high. We're also labeling the vertical axis as "Bytes" for clarity. The DEF lines define each of our RRD values and name them appropriately for later use in this graph, and use the LAST value for each so that we get the most current info.

The HRULE directive is one we haven't looked at before, it simply draws horizontal lines on the output graph at specified values. Here we're drawing HRULES at 192KB intervals (the thruput limit of a T1 in K) and coloring them blue. The idea of the HRULES is to easily and quickly see exactly how much of the T1 total bandwidth is being consumed by the multilink traffic.

Then we define the AREAs and LINEs. I'm only graphing out the Multilink and FastEthernet traffic, because if things are working properly the thruput of the FastEther should always match the thruput of the multilink, therefore we use an AREA for the MultiLink and a LINE1 for the FastEther, and if things are working properly you should always just see the FastEther lines outlining the top of the MultiLink area. It makes for quick reading. Please note that the order in which I've specified them is very important because each line/area is drawn in the order that you specified it. In this way, if the MultiLink Out area is larger than the MultiLink In, you'll never see the In values, but you will see the lines on top. You'll notice this in the output graph.

Finally, I use some GPRINT statements to put the thruput values of each individual T1 on the output graph. I choose not to actually graph these values because the graph just got way too crowded. In this case, we really don't care what the thruput of each T1 really is anyway, we just want to know that they are all roughly passing the same amount of traffic which means the MultiLink is working properly.

Now, I drop both the script for gathering the data and graphing the data into cron:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /home/benr/RRD/WAN-RRD/wan-update.pl
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /home/benr/RRD/WAN-RRD/make_graph.sh
```

Figure 8. Crontab Invocation of Updating and Graphing

After letting it run for awhile, we look at the graph and the following is the result.

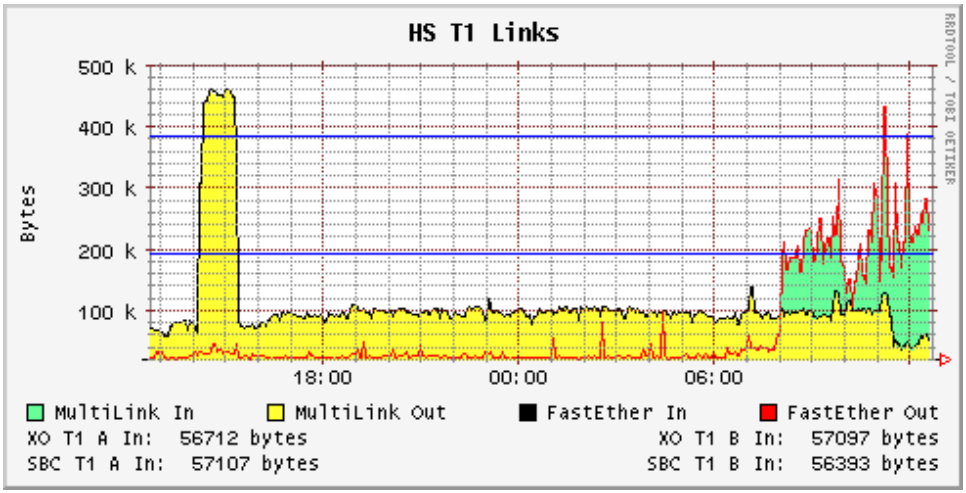

Figure 9. 1 Day Bandwidth Utilization Graph

In the above output graph you can see clearly everything we want to know about our MultiLinked T1's. This the output as seen on a Monday morning so you notice that over the weekend there was very little traffic coming into our network across the T1's, but there was a fairly steady amount of traffic going out (apparently this is Microsoft Actve Directory traffic). Because of the order in which we specified our AREA statements you don't see the green AREA (MultiLnk In) at all, but you do see the FastEtherOut which is drawn ontop of both AREAs. We could fix this by reordering the AREAs and LINEs but, as you can see, during the week the traffic going out far exceeds the traffic coming in, making for better graphing.

Viewing Historical Data

Continuing from our last example, lets create some other graphs for historical purposes. You'll recall that when we created the RRD we specified two different RRA's: LAST (for the most current data) and AVERAGE (for hourly averages). Now that I've got acouple days worth of data in my RRD I want to look at the historical graphs. We'll do this by creating a duplicate graphing script as we did above, but we'll add an extra argument to the **rrdtool graph** command: **--start** and optionally **--end**.

Lets just look at the example. The following graph generation script is exactly the same as the one we created in the last section, but I've added the **--start** argument and changed the RRA's specified in both the DEFs and GPRINTs from LAST to AVERAGE to ensure we get the right values.

```
#!/bin/bash

rrdtool graph wanoutput-hist.png -a PNG \
--title="HS T1 Links Hourly Averages" --vertical-label "Bytes" \
--height 150 \
--start "-1month" \
'DEF:s00in=wan-thruput.rrd:Serial00_In:AVERAGE' \
'DEF:s01in=wan-thruput.rrd:Serial01_In:AVERAGE' \
'DEF:s10in=wan-thruput.rrd:Serial10_In:AVERAGE' \
'DEF:s11in=wan-thruput.rrd:Serial11_In:AVERAGE' \
'DEF:fe20in=wan-thruput.rrd:FastEth20_In:AVERAGE' \
'DEF:mlin=wan-thruput.rrd:MutliLink_In:AVERAGE' \
'DEF:s00out=wan-thruput.rrd:Serial00_Out:AVERAGE' \
'DEF:s01out=wan-thruput.rrd:Serial01_Out:AVERAGE' \
'DEF:s10out=wan-thruput.rrd:Serial10_Out:AVERAGE' \
'DEF:s11out=wan-thruput.rrd:Serial11_Out:AVERAGE' \
'DEF:fe20out=wan-thruput.rrd:FastEth20_Out:AVERAGE' \
'DEF:mlout=wan-thruput.rrd:MutliLink_Out:AVERAGE' \
'HRULE:193000#0000ff' \
'HRULE:386000#0000ff' \
'HRULE:579000#0000ff' \
'HRULE:772000#0000ff' \
'AREA:mlin#66FF99:MultiLink In' \
'AREA:mlout#FFFF33:MultiLink Out' \
'LINE1:fe20in#000000:FastEther In' \
'LINE1:fe20out#ff0000:FastEther Out' \
'GPRINT:s00in:AVERAGE:XO T1 A In\: %6.0lf bytes ' \
'GPRINT:s01in:AVERAGE:XO T1 B In\: %6.0lf bytes\j' \
'GPRINT:s10in:AVERAGE:SBC T1 A In\: %6.0lf bytes' \
'GPRINT:s11in:AVERAGE:SBC T1 B In\: %6.0lf bytes\j'
```

Figure 10. 1 Month Historical Graph Creation Script

Using the graph arguments **--start** and **--end** we can specify the window of time the graph displays. By default (no start/end arguments) the graph starts 24hours ago, and ends "now", so that we always see one full day of values. If we want to see a larger period of time we can change the start time to something more useful. Generally you won't need to use the end argument because you want it to end now which is the default, but it

works just like the start argument does in case you wanted to, for instance, create a historical graph just for the month of December. Start and End times are specified in forms accepted by the UNIX **at** command, therefore you can specify absolute seconds since the Epoch or "Jan 10" as a specific date. You can also use more useful relative times such as "yesterday", "-1month" (One Month Ago), "-2weeks", "-1year", etc. Check out the rrdfetch man page for more details. In our graph above, I've decided to give 1 month of historical data. Below is the output.

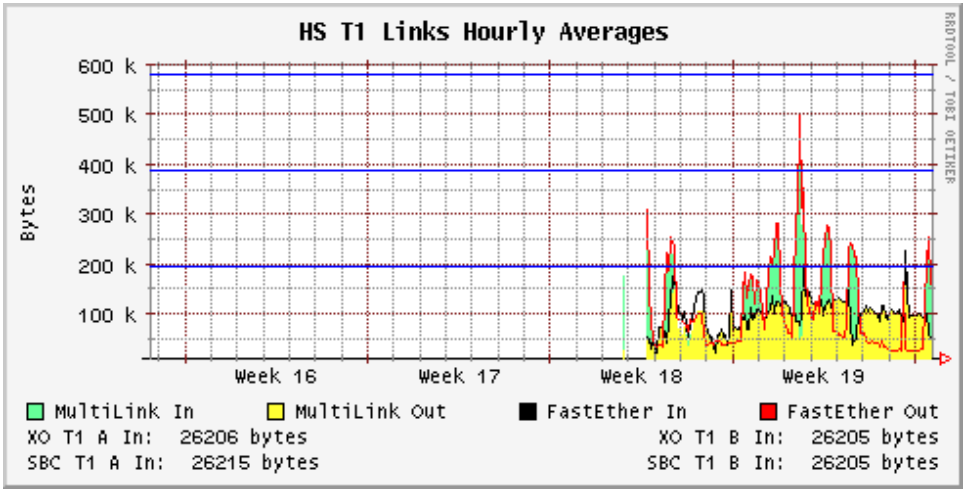

Figure 11. Generated 1 Month Bandwidth Graph

Using a script like the one above can be easily duplicated or modified in order to show almost any range of statistical data. Depending how many RRAs you created you can get some very granular details.

Conclusions

At first glance RRDtool is extremely frustrating and needlessly complicated, but most of this sentiment comes from a long time familiarization with simpler tools such as MRTG and a personal struggle with RRDtools unique syntax. However, once these hurdles can be overcome the benefits of RRDtool over almost any other tool available become immediately apparent. RRDtool embraces a "work smarter, not harder" approach once it's understood, even though it seems a lot harder than smarter in the end. Being able to have absolute control over your graph output, being able to consolidate 10+ MRTG graphs into 1 RRDtool graph , and having unlimited ability to get exactly the values you want and even manipulate them before being recorded (thanks to scripting) make RRDtool the power monitoring tool of choice.

Hopefully this tutorial has helped you get over the basic hurdles that RRDtool throws in your way and provides a gateway for better understanding of the tool. For more information, including other tutorials, man pages, examples, updates, and to get the source yourself, please visit the RRDtool website [http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/]. And if you like RRDtool, remember to show Tobi some love [http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/license.html].