

Monitoring a SAN with MRTG

The Storage Area Network is a relative newcomer to the SysAdmin's toolbox and with it brings a plethora of benefits. Unfortunately it also brings complexity.

We used to be familiar with disk arrays connected by thick SCSI cables, and we were confident that as long as we stuck to seven or less devices to a SCSI bus, we should receive adequate performance.

With the advent of SANs, those days are effectively over. SAN technology can potentially connect a server to hundreds or even thousands of storage devices via a single fibre pair. Similarly, a single host with multiple Host Bus Adaptors (HBAs) can generate a huge amount of cross SAN traffic, potentially causing contention on shared devices.

The flow of traffic will need to be managed as the SAN grows, but before the traffic can be effectively managed, we must be able to monitor activity.

We very often balance traffic across multiple HBAs for performance and redundancy. Thus, when considering the requirements for a SAN monitor it is important to consider that an "edge node" (a device on the outer periphery of the SAN) does not relate to a host, or a storage array, but to an HBA.

This considered, monitoring the SAN starts to become a headache – with multiple HBA's and multiple paths through the SAN from host to storage array.

I recently had a potential performance issue at a client site where multiple Solaris hosts were accessing a single EMC Symmetrix (via a series of Brocade switches). It was suspected that the activities of one or more of the hosts were negatively affecting the others, including a very important production system.

A more permanent monitoring solution was planned, but unlikely to be implemented within a month, and thus began this project, which gave us visibility of the SAN within a few hours of its' inception.

I chose Perl and PHP to implement this: Perl because of its' ability to handle complex data structures, and PHP for its' easy integration into the Apache webserver. With hindsight, I guess we should have stuck to one language, however – this was developed in a hurry. Perhaps that will be a job for the future.

Introducing MRTG

MRTG (Multi-Router Traffic Generator) is a data gathering and charting tool that has been written by Tobias Oetiker. As per its' name, it was originally developed to monitor traditional LAN and WAN devices.

Googling upon the MRTG website, it became clear that MRTG could do almost everything that we required. Given the correct SNMP configuration for the switches, it can query the fibrechannel statistics and graph each port on the switch.

MRTG also allows mathematical operations on the SNMP gathered data. This is a key enabler for this project, as it allows for multiple ports (potentially on different switches) to be aggregated together to compute the overall throughput for any given host into the SAN.

The only problem was that it was clear that as the number of switches, (and hence ports) monitored by the toolkit increased, the MRTG configuration file would rapidly become unmanageable.

Furthermore, it was clear that although MRTG does a great job of generating all the necessary graphs and HTML to our specification, we needed several layers of abstraction in order to rapidly "drill down" into the data when troubleshooting.

This data should all be easily accessible by a browser, and so it became clear that the visualisation layer would have to sit between the Apache server and the raw data to help the user interpret the results.

To solve these challenges, two programs were written: the configurator and the visualiser.

The monitoring host was to be an existing Sun E450, running Solaris 9, with the Apache webserver and Perl (as bundled by Sun). The latest version of PHP and MRTG were then installed (along with the required libraries – most of which are usefully available pre-packaged on <http://www.sunfreeware.com>).

Tobias has included some excellent documentation on the MRTG website, so we'll not discuss the details of installing MRTG here – once you've got the required libraries, it is an easy install.

To assist with the example given in this article, here is a simplified diagram of a fictitious SAN setup:

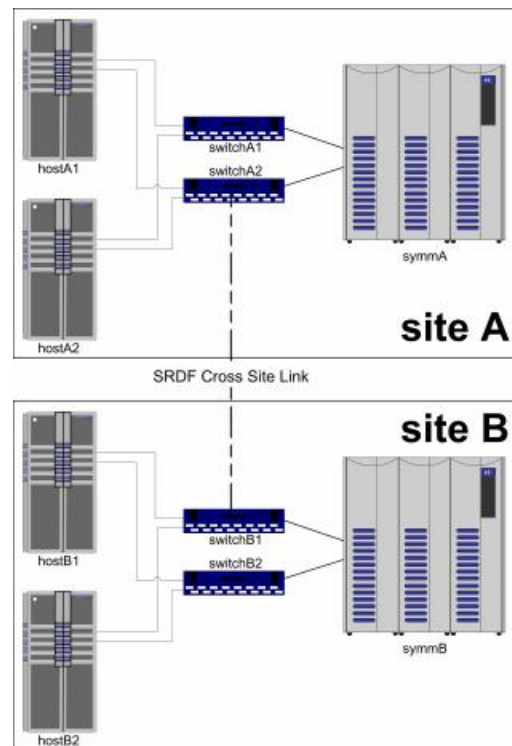


Figure 1 – Example configuration

switchA1 port assignments:

Port 7 symmA, FA#0
Port 1 hostA1, HBA#0
Port 4 hostA2, HBA#0

switchA2 port assignments:

Port 7 symmA, FA#1
Port 1 hostA1, HBA#1
Port 12 hostA2, HBA#1
Port 15 switchB1, port 15

switchB1 port assignments:

Port 3 symmB, FA#0
Port 1 hostB1, HBA#0
Port 9 hostB2, HBA#0
Port 15 switchA1, port 15

switchB2 port assignments:

Port 3 symmB, FA#1
Port 1 hostB1, HBA#1
Port 9 hostB2, HBA#1

Configuration

The first part of the exercise is to translate the actual switch configuration into the data structure at the head of the configurator program. This has already been done in Listing 1 for the example setup.

```
#!/usr/bin/perl -w

# SAN To MRTG Configurator
# Mike Scott      : mikes@hindsight.it
# Version         : 1.0
# Date           : Feb 2003

#####
# WARNING - THIS DATASTRUCTURE IS CASE SENSITIVE
# Make sure that hostnames match exactly...
#####
%configuration= (
  "sitea" => {
    "switch1" => {
      #
      PORT => [HOSTNAME,      HBA NAME      ]
      "01" => ["hostal",      "fcaw0"    ],
      "04" => ["hosta2",      "fcaw0"    ],
      "07" => ["symma",       "FA-2"     ]
    },
    "switcha2" => {
      "01" => ["hostal",      "fcaw1"    ],
      "12" => ["hosta2",      "fcaw1"    ],
      "07" => ["symma",       "FA-1"     ],
      "15" => ["switchb1",    "port15"   ]
    },
  },
  "siteb" => {
    "switchb1" => {
      "01" => ["hostb1",      "fcaw0"    ],
      "09" => ["hostb2",      "fcaw1"    ],
      "03" => ["symmb",       "FA-1"     ],
      "15" => ["switchal",    "port15"   ]
    },
    "switchb2" => {
      "06" => ["hostb1",      "fcaw1"    ],
      "09" => ["hostb2",      "fcaw1"    ],
      "03" => ["symmb",       "FA-2"     ]
    },
  },
);

# Some globals...
$MRTGDIR="/export/data/MRTG";
# MRTG html directory
$HTMLDIR="$MRTGDIR/html";
# MRTG config directory
$CFGDIR="$MRTGDIR/cfg";
# MRTG log directory
$LOGDIR="$MRTGDIR/logs";

#####
# Routine to generate the MRTG configuration per port
sub brocade port ($$) {
  # These two values are the OIDs from the Brocade MIB that specify
  # swFCPortTxWords and swFCPortRxWords. They represent the number
  # of 4 byte Fibrechannel words seen passing through any given port
  # (whose port number is appended to the end of the OID)
  my $OID INPUT="1.3.6.1.4.1.1588.2.1.1.1.6.2.1.11.";
  my $OID OUTPUT="1.3.6.1.4.1.1588.2.1.1.1.6.2.1.12.";
  my ($port,$switch)=@;

  # Physical port 0 is referred to as port 1 in the
  # SNMP information
  $port++;

  # Multiply the SNMP values (swFCPortTxWords/swFCPortRxWords) by 4 to
  # get the number of bytes.

  # Also notice, the :1 - we are using SNMPv1 - I noticed that the
  # Brocade 2800 appears to support v2, but our 12000 does not.
  return "(
    ${OID INPUT}${port}\&${OID OUTPUT}${port}:public@${switch}:::1 * 4 )"
}

#####
# A quick and easy routine to check if a directory exists, if not
# then create it.
sub checkAndMkdir ($) {
  my ($dir)=@;

  if ( ! -d $dir ) {
    print "Warning: Creating directory $dir\n";
    mkdir($dir,0755) || die "Error: Could not create $dir\n";
  }
}

#####
# Main Program

checkAndMkdir("$CFGDIR");
checkAndMkdir("$HTMLDIR");
checkAndMkdir("$HTMLDIR/group");
checkAndMkdir("$LOGDIR");
checkAndMkdir("$LOGDIR/group");

# Loop once per group (which is a general term for "site" in our context)
foreach $group (keys %configuration) {

  # Clear the %hosts cache
  my %hosts = ();
  print "Generating $group.cfg file...\n";

  # take a copy of the group configuration
  # Not particularly efficient, but it makes referencing the complex
  # data structure easier...
  my %conf=%{ $configuration{$group} };

  # First we need to invert the config, so we are
  # looking at it from a host-centric view, rather than
  # the switch-centric view...
  foreach $switch (keys %conf) {

    # Loop once per defined switch port
    foreach $port (sort keys %{$conf{$switch}}) {

      # Get the hostname and HBA from the config
      ($hostname,$interface)=@{$conf{$switch}-
    >{$port}};

      # Generate a hash of hosts in the SAN
      # each hash element is a list of switches and
      ports
      # that the machines' HBAs are connected to
      push(@{$hosts{$hostname}},"$switch:$port");
    }

    # Create the HTML and log directories, if they don't already
    # exist
    checkAndMkdir("$HTMLDIR/group/$group");
    checkAndMkdir("$HTMLDIR/group/$group/img");
    checkAndMkdir("$LOGDIR/group/$group");

    # Each group has its' own MRTG config file to be processed separately
    open(CFG,">$CFGDIR/$group.cfg") || die "ERROR: Cannot open
  $CFGDIR/$group.cfg\n";
    print CFG "HtmlDir: $HTMLDIR/group/$group/\n";
    print CFG "ImageDir: $HTMLDIR/group/$group/img/\n";
    print CFG "LogDir: $LOGDIR/group/$group/\n";
    # If a global config file exists, then include it into the
    # generated config
    if ( -f "$CFGDIR/global.inc" ) {
      print CFG "Include: $CFGDIR/global.inc\n";
    }

    # Generate the MRTG config for each host in the SAN
    foreach $hostname (sort keys %hosts) {
      @host aggregate=();
      @host hbalist=();

      # Generate the config for each HBA port
      foreach $hba (@{$hosts{$hostname}}) {
        ($switch,$port)=split(/:/,$hba,2);

        # Get the HBA interface name
        (undef,$interface)=@{$conf{$switch}->{$port}};

        # Generate the simple configuration for the HBA
        print CFG "Target[${switch} port${port}]:
      ".&brocade port($port,$switch)."\n";
        print CFG "Title[${switch} port${port}]:
      Throughput for $switch:$port ($hostname:$interface)\n";
        print CFG "PageTop[${switch} port${port}]:
      <H1>Throughput for $switch:$port ($hostname:$interface)</H1>\n";

        # Store the MRTG config for this HBA - this will
        be used
        # Once the loop is finished to generate the
        aggregate config
        push(@host aggregate,&brocade port($port,$switch));
        push(@host hbalist,"<a
      href=${switch} port${port}.html><img src=img/${switch} port${port}-day.png height=100
      width=320 alt=\"${switch}:${port} to
      ${hostname}:${interface}\"><br><center>${switch}:${port} to
      ${hostname}:${interface}</center></a>");
      }

      # Take all the HBA config cached in @host aggregate and add
      them together
      print CFG "Target[$hostname]:
      ".join(" ",@host aggregate)."\n";
      print CFG "Title[$hostname]: Aggregate Throughput for
      $hostname\n";
      print CFG "PageTop[$hostname]: <H1>Aggregate Throughput for
      $hostname</H1>\n";
      print CFG "PageFoot[$hostname]: <hr><table>";

      # A nice pretty table at the foot of the aggregate page to
      # contain thumbnails and links to the individual HBA graphs,
      # which we stored in @host_hbalist
      my $count=0;
      my $GRIDWIDTH=2;
      foreach (@host_hbalist) {
        if ($count%$GRIDWIDTH==0) {
          print CFG "<tr>";
        }
        print CFG "<td>$</td>";
        if ($count % $GRIDWIDTH==( $GRIDWIDTH-1 )) {
          print CFG "</tr>";
        }
        $count++;
      }

      while ($count % $GRIDWIDTH gt 0) {
        print CFG "<td>&nbsp;</td>";
        $count++;
      }

      print CFG "</tr></table>\n";
    }
    close(CFG);
  }
}

```

```

#####
# Main Program

checkAndMkdir("$CFGDIR");
checkAndMkdir("$HTMLDIR");
checkAndMkdir("$HTMLDIR/group");
checkAndMkdir("$LOGDIR");
checkAndMkdir("$LOGDIR/group");

# Loop once per group (which is a general term for "site" in our context)
foreach $group (keys %configuration) {

  # Clear the %hosts cache
  my %hosts = ();
  print "Generating $group.cfg file...\n";

  # take a copy of the group configuration
  # Not particularly efficient, but it makes referencing the complex
  # data structure easier...
  my %conf=%{ $configuration{$group} };

  # First we need to invert the config, so we are
  # looking at it from a host-centric view, rather than
  # the switch-centric view...
  foreach $switch (keys %conf) {

    # Loop once per defined switch port
    foreach $port (sort keys %{$conf{$switch}}) {

      # Get the hostname and HBA from the config
      ($hostname,$interface)=@{$conf{$switch}-
    >{$port}};

      # Generate a hash of hosts in the SAN
      # each hash element is a list of switches and
      ports
      # that the machines' HBAs are connected to
      push(@{$hosts{$hostname}},"$switch:$port");
    }

    # Create the HTML and log directories, if they don't already
    # exist
    checkAndMkdir("$HTMLDIR/group/$group");
    checkAndMkdir("$HTMLDIR/group/$group/img");
    checkAndMkdir("$LOGDIR/group/$group");

    # Each group has its' own MRTG config file to be processed separately
    open(CFG,">$CFGDIR/$group.cfg") || die "ERROR: Cannot open
  $CFGDIR/$group.cfg\n";
    print CFG "HtmlDir: $HTMLDIR/group/$group/\n";
    print CFG "ImageDir: $HTMLDIR/group/$group/img/\n";
    print CFG "LogDir: $LOGDIR/group/$group/\n";
    # If a global config file exists, then include it into the
    # generated config
    if ( -f "$CFGDIR/global.inc" ) {
      print CFG "Include: $CFGDIR/global.inc\n";
    }

    # Generate the MRTG config for each host in the SAN
    foreach $hostname (sort keys %hosts) {
      @host aggregate=();
      @host hbalist=();

      # Generate the config for each HBA port
      foreach $hba (@{$hosts{$hostname}}) {
        ($switch,$port)=split(/:/,$hba,2);

        # Get the HBA interface name
        (undef,$interface)=@{$conf{$switch}->{$port}};

        # Generate the simple configuration for the HBA
        print CFG "Target[${switch} port${port}]:
      ".&brocade port($port,$switch)."\n";
        print CFG "Title[${switch} port${port}]:
      Throughput for $switch:$port ($hostname:$interface)\n";
        print CFG "PageTop[${switch} port${port}]:
      <H1>Throughput for $switch:$port ($hostname:$interface)</H1>\n";

        # Store the MRTG config for this HBA - this will
        be used
        # Once the loop is finished to generate the
        aggregate config
        push(@host aggregate,&brocade port($port,$switch));
        push(@host hbalist,"<a
      href=${switch} port${port}.html><img src=img/${switch} port${port}-day.png height=100
      width=320 alt=\"${switch}:${port} to
      ${hostname}:${interface}\"><br><center>${switch}:${port} to
      ${hostname}:${interface}</center></a>");
      }

      # Take all the HBA config cached in @host aggregate and add
      them together
      print CFG "Target[$hostname]:
      ".join(" ",@host aggregate)."\n";
      print CFG "Title[$hostname]: Aggregate Throughput for
      $hostname\n";
      print CFG "PageTop[$hostname]: <H1>Aggregate Throughput for
      $hostname</H1>\n";
      print CFG "PageFoot[$hostname]: <hr><table>";

      # A nice pretty table at the foot of the aggregate page to
      # contain thumbnails and links to the individual HBA graphs,
      # which we stored in @host_hbalist
      my $count=0;
      my $GRIDWIDTH=2;
      foreach (@host_hbalist) {
        if ($count%$GRIDWIDTH==0) {
          print CFG "<tr>";
        }
        print CFG "<td>$</td>";
        if ($count % $GRIDWIDTH==( $GRIDWIDTH-1 )) {
          print CFG "</tr>";
        }
        $count++;
      }

      while ($count % $GRIDWIDTH gt 0) {
        print CFG "<td>&nbsp;</td>";
        $count++;
      }

      print CFG "</tr></table>\n";
    }
    close(CFG);
  }
}

```

Listing 1 – The configurator

The raison d'être for this program is to allow a SAN configuration to be specified in a compact human readable format. This ensures that it can

be updated easily. The configurator will then take the SAN configuration and generate suitable MRTG configuration files.

The first part of the program is clearly the data structure that specifies the SAN. This data structure could be described as "switch-centric" – this makes it very easy to accurately populate the initial values, as it can be compared directly with the output of the Brocade "switchShow" command, which will display which ports of the fibrechannel switch are in use (of course, the Brocade command will show the World-Wide Numbers of the connected devices, which may require translation).

The data structure could easily be hived off to a separate file, or even integrated with a web front end to make updates even easier – however for the sake of simplicity, it has been left integrated with the Perl program.

The remainder of the program consists of several "foreach" loops which will firstly invert the data structure so that that it can be later interpreted as "host-centric", and per-host aggregate formulae constructed.

When executed, the program proceeds to process the configuration data structure and correlates each HBA with its' respective host. It then produces the MRTG configuration files (one per group).

Listing 2 shows an example output from the configurator tool. For brevity, we have included only the first three targets of the "sitea.cfg" file.

```
HtmlDir: /export/data/MRTG/html/group/sitea/
ImageDir: /export/data/MRTG/html/group/sitea/img
LogDir: /export/data/MRTG/logs/group/sitea/
Include: /export/data/MRTG/cfg/global.inc
Target[switchal port01]: (
1.3.6.1.4.1.1588.2.1.1.1.6.2.1.11.02&1.3.6.1.4.1.1588.2.1.1.1.6.2.1.12.02:public@swit
chal:::1 * 4 )
Title[switchal port01]: Throughput for switchal:01 (hostal:fcaw0)
PageTop[switchal port01]: <H1>Throughput for switchal:01 (hostal:fcaw0)</H1>
Target[switcha2 port01]: (
1.3.6.1.4.1.1588.2.1.1.1.6.2.1.11.02&1.3.6.1.4.1.1588.2.1.1.1.6.2.1.12.02:public@swit
cha2:::1 * 4 )
Title[switcha2 port01]: Throughput for switcha2:01 (hostal:fcaw1)
PageTop[switcha2 port01]: <H1>Throughput for switcha2:01 (hostal:fcaw1)</H1>
Target[hostal]: (
1.3.6.1.4.1.1588.2.1.1.1.6.2.1.11.02&1.3.6.1.4.1.1588.2.1.1.1.6.2.1.12.02:public@swit
chal:::1 * 4 )+(
1.3.6.1.4.1.1588.2.1.1.1.6.2.1.11.02&1.3.6.1.4.1.1588.2.1.1.1.6.2.1.12.02:public@swit
cha2:::1 * 4 )
Title[hostal]: Aggregate Throughput for hostal
PageFoot[hostal]: <H1>Aggregate Throughput for hostal</H1>
PageFoot[hostal]: <tr><table><tr><td><a href=switchal port01.html><img
src=img/switchal port01-day.png height=100 width=320 alt="switchal:01 to
hostal:fcaw0"><br><center>switchal:01 to hostal:fcaw0</center></a></td><td><a
href=switcha2 port01.html><img src=img/switcha2 port01-day.png height=100 width=320
alt="switcha2:01 to hostal:fcaw1"><br><center>switcha2:01 to
hostal:fcaw1</center></a></td></tr></table>
```

Listing 2 – excerpt from "sitea.cfg"

Notice that if a "global.inc" file exists in the configuration directory at the time that the configurator is executed, it will be included into the group configuration files via a MRTG "Include" directive. This allows us to have a file which can specify options that are common to all groups. An example of which is given in Listing 3. Also notice the use of the AddHead directive in the "global.inc" file to include a CSS stylesheet to ensure that all pages have the same look as the rest of the website.

```
-----
#-- Global Defaults
#-----
# Server is a quad processor E450, with two instances of MRTG running - Forks
# set to 2 to make better use of the processors
Forks: 2
IconDir: /icons
WriteExpires: Yes
Refresh: 300
Options[_]: noinfo, growright, printrouter, pngdate, nobanner, unknaszero
MaxBytes[_]: 134217728
Unscaled[_]: dwmy
LegendI [_]: in&nbsp;
LegendO [_]: out&nbsp;
Legend1 [_]: in&nbsp;
Legend2 [_]: out&nbsp;
YLegend[_]: port throughput
WithPeak[_]: dwmy
AddHead[_]: <LINK REL="STYLESHEET" HREF=/standard.css>
XSize[_]: 600
YSize[_]: 300
kilo[_]: 1024
#-----
```

Listing 3 – global.inc

Once the configuration files have been generated, MRTG can be started according to the instructions on the MRTG website. For this, I chose to run it from cron every five minutes (preferably by a non-privileged user). Listing 4 shows an example crontab extract.

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /usr/local/mrtg-2/bin/mrtg
/export/data/mrtg/cfg/sitea.cfg --logging=/export/data/mrtg/log/sitea.log
1,6,11,16,21,26,31,36,41,46,51,56 * * * * /usr/local/mrtg-2/bin/mrtg
/export/data/mrtg/cfg/siteb.cfg --logging=/export/data/mrtg/log/siteb.log
```

Listing 4 – sample crontab entries

Visualisation

By this point, you should have MRTG up and running. It should be generating HTML pages with PNG graphs, which is very useful, but very difficult to navigate.

The visualiser is a short piece of PHP code that will scan the group directories and present thumbnails in a logical and structured manner. Graphs are presented of the aggregate graphs only – meaning that the clutter of the individual HBA graphs will be masked from the user.

When the visualiser presents a page of thumbnails, it links each thumbnail back to the MRTG generated HTML page for the host's aggregate graphs. The user can then expand the selection simply by clicking on the thumbnail graph to obtain a better view of what is going on with that machine.

Figure 2 shows an example of the group view, showing all hosts and storage devices. Notice also that switchb1 and switcha1 are listed – these graphs represent the inter-switch links.

SAN Monitoring

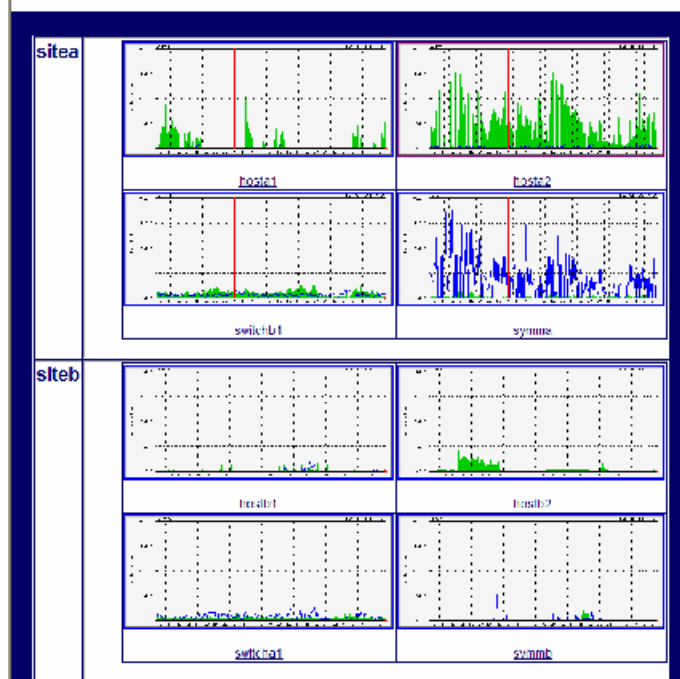


Figure 2 – the visualiser - group view

The visualiser code should be installed in your \$HTMLDIR, as "index.php", and the apache web server configured with the directive "DirectoryIndex index.php" in order that it is picked up automatically when the browser requests a directory index.

```
<head></head><body>
<LINK REL="STYLESHEET" HREF=/standard.css>
<TITLE>SAN Monitoring</title>
<center>
<table width=100% cellspacing=0>
<tr valign=center><td><h1>SAN Monitoring</h1></td></tr></table></center>
<table border=1 >
<?php
# SAN To MRTG Visualiser
# Mike Scott : mike@hindsight.it
# Version : 1.0
# Date : Feb 2003

# Globals
$HTMLDIR="/export/data/mrtg/html";

#####
# Check input parameters (if any)

# browseBy=[d|w|m|y|none] - day,week,month,year or no thumbnails
```


Each HBA thumbnail can similarly be expanded by clicking on the graph thumbnail to give a detailed report of the activities of that individual adapter. Figure 4 shows an example of this.

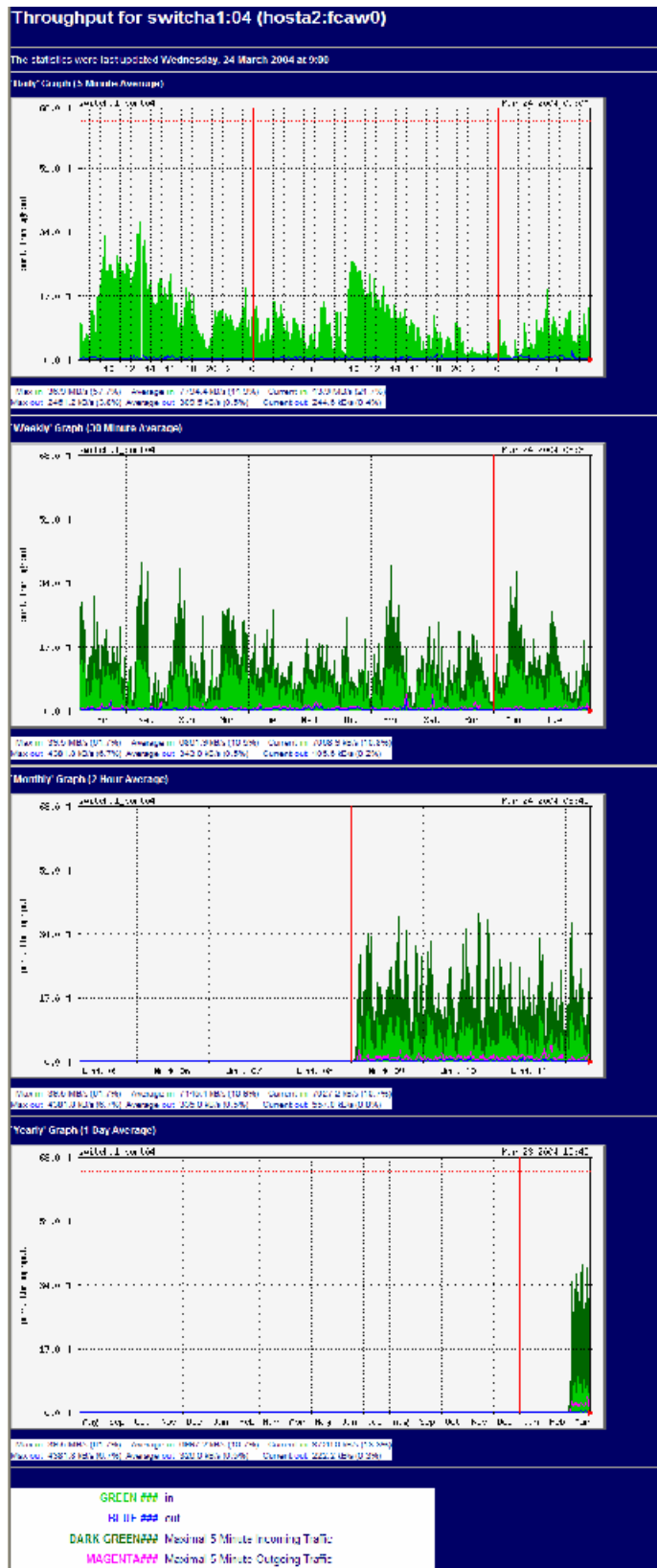


Figure 4 – the visualiser – HBA view

Conclusion

Enterprise monitoring packages such as BMC Patrol and TeamQuest are very good at what they do, but occasionally we need to look at very specific sets of data for a short period of time while performance troubleshooting. Very often, the easiest way to achieve this is to have a package such as MRTG available in your toolbox that can quickly be deployed for an ad-hoc request.

In this instance MRTG allowed us to very quickly visualise the entire SAN. We identified key performance facts that we were not previously aware of:

- relative load on each individual Symmetrix from each server,.
- correlation of SAN activity against specific events (e.g. significant data loading operations that were causing concern for contention on the Symmetrix arrays).
- Identification of times of significant activity, allowing us to feedback to the application owners with time based data in order to reschedule I/O intensive applications to a quieter time.

Very often we wish to monitor critical servers. In this case, it is often difficult to justify significant changes that may impact production. Monitoring the SAN from the switches via SNMP is perceived as a very non-intrusive method (and certainly easier than deploying software agents to all SAN-connected hosts). This solution would likely be approved by even the strictest change-management policies.

MRTG is an extremely flexible tool that enabled us to rapidly begin monitoring the SAN environment to analyse a specific problem. It is still being used today at the client site and is considered to be a valuable tool in performance monitoring. Tobias Oetiker and Dave Rand have done a tremendous job in developing this package, and it certainly deserves a closer inspection if you haven't already looked at it.

The code presented in this article is an example of a rapidly developed application, and could certainly be improved upon. It is, however a good example of what may be achieved by taking an existing generic application, and extending it with tools like PHP and Perl to suit a very specific set of requirements.

An archive of these scripts, including more detail on the install instructions than was possible to include in this article is available at <http://hindsight.it/san/>

Mike Scott is the director of Hindsight IT Ltd, a small Solaris consultancy based in Central Scotland. He has been working in the North East and the central belt for the last ten years, specialising in systems management with a keen interest in security and performance management. He can be contacted at syadmin@hindsight.it

References

- [1] Tom Clark, "Designing Storage Area Networks", Addison-Wesley, 2000
- [2] Chris Beauchamp and Josh Judd, "Building SANs with Brocade", Syngress, 2001
- [3] Tobias Oetiker, Dave Rand, Multi Router Traffic Grapher, available at <http://people.ee.ethz.ch/~oetiker/webtools/mrtg/>