Monitoring the World with NetBSD

Alan Horn
Inktomi Corp
Alan.Horn@inktomi.com

Jennifer Davis
California Institute of Technology
Jennifer.Davis@caltech.edu

Abstract

Through presenting a set of guidelines and freeware monitoring tools, we hope to prevent your enterprise from experiencing embarrassing and costly mistakes. Part of building any system, whether from a fresh start, or adding to a preexisting architecture, requires this kind of planning, although the depth depends on the complexity of your environment. This will help prevent system degradation and public embarrassment as well as improve perceived system performance.

1. Introduction

System environments are becoming more complex as customer needs and requirements increase. Keeping up with competitors in this negative market requires that money is not wasted on trivial matters. As budgets are tightened, and hiring additional IT techs is frozen, managing the growing spider web of systems and networks becomes more difficult. Time is consumed by tracking down problems, security patches, and basic system management. With an effective monitoring solution, administrators can free up their time for more important tasks.

1.1 Define the problem.

In the standard environment, monitoring comes last after purchasing adequate equipment, setting up the system and required services, and making the system live.

Sometimes, monitoring is never considered, as the overworked administrator is charged with all of the previous tasks, little resources to accomplish them, and the demand that it all be finished yesterday. Monitoring becomes important when the company financial controller (Mr. Smiley) realizes how much money the company has lost because of unforeseen outages in the last year. This is when the IT department is charged with setting up a low cost monitoring solution.

1.2 What is monitoring?

What do we mean by our desire to set up a monitoring solution? If we reference http://www.webster.com

v. mon·i·tored, mon·i·tor·ing, mon·i·tors [Latin, from monre, to warn]
v. tr.
1. To check the quality or content of (an electronic audio or visual signal) by means of a receiver.
2. To check by means of an electronic receiver for significant content, such as military, political, or illegal activity: monitor a suspected criminal's phone conversations.
3. To keep track of systematically with a view to collecting information: monitor the bear population of a national park; monitored the political views of the people.
4. To test or sample, especially on a regular or ongoing basis: monitored the city's drinking water for impurities.
5. To keep close watch over; supervise: monitor an examination.
6. To direct.

From this definition, we can define a comprehensive solution to our problem. Monitoring comprises regularly sampling some sort of content, systematically tracking the state of that content, and warning the appropriate parties when necessary.

1.3 What should we be monitoring?

Determining what should be monitored is a decision that should be made by analyzing individual environments. For critical out facing machines, monitoring the world may be the only solution. For stand-alone work machines, the system may not need to be monitored at all. The difference depends on what people consider important, why they consider it important, and who those people are.

The main types of monitoring are uptime or availability, performance, and security. The goals of monitoring are to make the job easier, more manageable, and efficient, and to fix problems before they are seen. If you work for a profit making company, ultimately you assist in 'increasing shareholder value'.

Monitoring is more than the world of bits and bytes. It can also involve the physical environment in which your systems live.

Monitoring should not replace redundancy of systems and services. Redundancy prevents complete outages allowing continued service, with possible degradation during a failure situation. Monitoring complements redundancy by alerting the IT department to fix the degraded state of services. For example, a RAID 5 disk array can suffer the loss of one disk, and still function. If the monitoring system alerts the system administrator immediately that a failure has occurred, then the disk can be replaced quickly without any loss of service or data on the array.

2.0 Selecting your tools.

The tools you select for monitoring should be dependable, stable, and consistent to the purpose to which you put them. As the rest of the company adopts the monitoring solution, the tools need to have a sufficiently rich set of features that provide flexibility to this expansion. The tools should be of a clean design, and readily understandable with a modicum of effort from others once your strategy has been implemented. For some, the tools must be quickly implemented as previous outages have made setting up a monitoring solution now a crisis.

2.1 One OS to rule them all

Although NetBSD is the authors' preferred OS, the techniques and applications discussed will work on other operating systems. The important factors for determination of OS are knowledge of OS, comfort levels at the basic levels of administration, integration into the existing environment, and standing political issues within your company.

There are several reasons to choose NetBSD over other operating systems. NetBSD provides a stable clean design and implementation and is well documented. It has a great network stack. It also has a comprehensive base Unix system with good analysis tools for simple monitoring, and readily available packages in the pkgsrc system. NetBSD runs on many platforms, as well as cheap commodity hardware. The final personal reason is personal comfort with the OS, and brand loyalty.

2.2 Pkgsrc system – an overview

Similar to the FreeBSD ports collection, the NetBSD pkgsrc system is a very good source of tools that is growing daily. Pkgsrc is not installed by default, but it is easily set up. Instructions for obtaining it are found in the references.

To see what packages are available in pkgsrc specifically for monitoring, check the net/, security/, and sysutils/ subdirectories.

```
$ cd /usr/pkgsrc/net
$ less */DESCR
```

This will show you the descriptions of every package within the net subsection.

To install a pkgsrc package (e.g. nocol) :

```
$ su
# cd /usr/pkgsrc/net/nocol
# make
# make install
```

Package binaries are typically installed in /usr/pkg/bin or /usr/pkg/sbin. Modify your $PATH environment variable as appropriate to include these new paths within your executable path.[1]

If you have problems when installing a package, contact the maintainer of the package. The person responsible for the maintenance of each individual package, is listed in the top-level Makefile in each pkgsrc package directory (e.g. /usr/pkgsrc/net/nocol/Makefile).

[1] Paths can be set on an individual per-user level, or you can modify /etc/csh.cshrc and /etc/profile to set the additional path for every user on the system.

3. Monitoring for availability

3.1 Definition

Availability is more than a system being up, and a service running. Availability means that individuals can get to what they need, when they need it, retrieve what it offers, and avail themselves of the service. For example, just because on start of Apache the "httpd started"message pops up, and the process logs show that http is up, and running, this does not indicate whether a user can actually reach a the document root, or any other page on the site. Another example, just because the Oracle ora_* processes are up and running, doesn't indicate whether that database is accepting connections and relaying the data needed.

3.2 Strategies

The best method of monitoring for availability, is to emulate as closely as possible the normal 'request operation'. A 'request operation' (RO) is how a user of the system, process on the system, or a process on another system accesses a specific service.

Examples of Access Tests for different services
web server - access the root document. (http://my.webserver.org)

name server – get the start of authority record (SOA) for a given zone, with additional tests confirming that the A records and PTR records for critical hosts are served correctly.

Critical web application – suite of tests, each element confirming a different stage or aspect of the web application is performing correctly.

Make informed decisions about the frequency of RO queries. By querying ROs too frequently, monitoring can induce additional load that degrades service performance. By not querying ROs enough, monitoring can miss outages that occur between your ROs. A good monitoring system will include tunable timing parameters such as these.

Depending on the tool chosen for each RO, tests are made on the exit status of the tool's run or a parse of the RO's output.

3.3 Tools

Monitoring focuses on the testing of a condition, and warning if that condition is not satisfied. One method of obtaining monitoring tools, is to subvert the function of a tool whose primary purpose is not monitoring.

Availability monitoring tools

ping – Ping is a basic function to test network connectivity

fping – Ping's big brother, fping will send ICMP connectivity checks to any number of hosts (specified on the command line or via an input file). It checks the hosts asynchronously with timeouts, meaning that it can be used in scripts very easily, and is generally a better choice than ping.

Snips (formerly known as nocol) – Snips is one of the 'uber-monitors'. It comes with a variety of testing tools to check against specific services. It has the ability to notify on alerts. It also has a simple control interface with four levels of alerting based on how often a check fails, with the alerts being configurable based on the levels. Snips contains built in monitors for:
ICMP ping
RPC portmapper
OSI ping
Ethernet load
TCP ports
Name server
Radius server
Syslog messages
Mailq
NTP
UPS (APC) battery
Unix host performance
BGP peers
SNMP variables
Data throughput

Nagios – Formerly known as netsaint, nagios is similar to snips in concept, but with additional features. Nagios has a very flexible alerting and notifications system, a nice GUI interface, and tactical display. It has a complete set of small binaries for standard service checks. Nagios has standard plugin check binaries for:
Dig disk space dns fping game hpjd http https ide-smart imap ldap load mrtg mrtgtraf mysql nagios nntp nt nwstat overcr pgsql ping pop procs radius real smtp snmp ssh swap tcp time udp ups users

vsz

There are also sample perl script checks for: breeze disk_smb flexlm ifoperstatus ifstatus ircd log netdns ntp oracle rpc sensors wave

Tkined – tkined is a TK application based on Scotty. It can be useful interactively for discovering networks, and live monitoring, but requires extensive customization to work in the background.

Lynx – Lynx is a simple text based web client. lynx-dump can grab raw documents in text for you to parse with additional code.

Wget – wget is a tool for grabbing both http and ftp files from servers.

ftp – The native NetBSD ftp client can be placed in non-interactive scripts to retrieve a particular dataset. The exit status of the command can be checked (0 means everything was successful).

Bigbrother – Bigbrother is another 'uber-monitor' similar to nagios. It's been around longer and has a far wider range of user-contributed plug-ins for monitoring lots of different services.

Bigsister – Bigsister is a clone of big brother developed to add different features, and also to improve performance by avoiding shell scripts.

Perl scripts using Net:: modules – With some creativity a perl script can be written to check for almost any network service. There are some limitations when testing crypto-based services.

Built-in software testing tools for a given device – For example on a Solaris system, A1000 raid array supporting software has a tool called 'healthck' which can be used to test for problems with the raid array. Other devices and packages will have tools that may be used similarly.

4. Monitoring for performance

4.1 Definition

Performance is harder to fully monitor as you may not have control over all machines involved in the transaction, or the network. To the user, acceptable performance generally means 'after I perform a specified action, the response occurs in a reasonable amount of time. For example, just because a web server's http process is showing up in process stats, this doesn't indicate anything about the fact that the server is taking 2 minutes to respond to each query. Another example, is email delivery. Just because sendmail is up and running, this doesn't give the administrator any idea if mail is getting processed, or whether time critical emails are getting delivered in time. Although the coverage from the system performing the actions to the server sitting in your data center may not be completely your responsibility, you can confirm that your responsibilities are functioning to their maximum performance.

4.2 Strategies

Along with ensuring the availability of a service, system, or network, there may be extended information you can or should gather about the performance of that availability. The information you should know or gather before setting up each individual monitor is the knowledge of the baseline performance i.e. normal operation of system or service. Also spend time predicting the failure modes and methods of degradation in performance for the object you are monitoring.

With this information, figure out the performance counters to monitor, and at what thresholds to alarm. You can choose to never alarm, and you will still have a body of historical information to analyze later.

At a minimum, any system providing a service should have the underlying system performance monitored; CPU load, disk space, I/O of all kinds. These basics will allow you to monitor system changes, which will provide data towards upgrading system resources, or the system itself as needed. Forewarned, you can kill out of control processes before they cause degraded performance. If you don't want to monitor these basic metrics, regularly take a few performance baselines so that in a crisis you have a record of what the system should look like. Individual services may also have metrics that you can monitor such as response time, resources used, etc.

System performance (and to a large extent service performance), cannot always easily be monitored across the network. SNMP is one popular solution, but it has drawbacks :

o Generally requires a complex daemon to run on a port.

o Susceptibleness to security issues[2].
o Overcomplication due to attempting to satisfy for all eventualities.
o The simple in SNMP (Simple Network Management Protocol) is not in reference to the configuration.

SNMP is probably the best choice for monitoring systems such as network switches and routers that do not have a feature rich operating system that you can access. SNMP is not the best choice for servers. Instead, use available local system tools, and either write a tight piece of code to send specific information back to the centralized monitoring host periodically or on demand, or avail yourself of a tool that performs this function.

Depending on the OS of the machine you are monitoring, different tools with varying ease of installation, and use will be available to you.

4.3 Tools

Sampling of tools for performance purposes
ps – With ps you can display process status on a host, and other useful information depending on the options available on your OS, such as cpu time, memory used by a process, owner of the process, etc.
df – df is helpful in determining filesystem usage.
uptime - system uptime and runq load
iostat – iostat reports information about I/O load.
vmstat - vmstat reports information about virtual memory. One of the best sources for determining what ails your server, cpu problems, excessive swapping, etc.
netstat – netstat provides you with various information about network stats. netstat –a provides information about all sockets. netstat –rn provides information about routing tables.
mrtg – mrtg is a simple tool for monitoring and graphing traffic load on network links
rrdtool – rrdtool is mrtg++, a round robin database tool developed based upon MRTGs graphing and logging features. Will display any time series. If you need to view data samples over time, this is the way to store it for analysis.
cricket – cricket is a rrdtool frontend focused on SNMP stats gathering for monitoring network hardware.
flowscan – flowscan is another rrdtool frontend using cflowd to gather Cisco flow data.
smokeping – smokeping is another rrdtool frontend that presents network latency in an MRTG style graph.
perl - Naturally, you can write your own tools in perl using the rich set of Net:: and other modules. Scripts can be written in shell, but you may also consider writing the glue scripts in perl too

[2] In the widely implemented SNMPv1, administrative relationships known as communities, are defined for SNMP entities. As long as you know the name of this community, you can access that particular SNMP community. This community name is used as a pseudo password to gain access to an SNMP device. To compound the problem, with every SNMP packet, the pseudo password is passed in clear text.

5. Monitoring for security

5.1 Definition

Beyond monitoring for availability and performance, you should monitor for security reasons. The material necessary to describe monitoring for security would provide the source for an entire paper by itself. If security was not included, our monitoring solution would not be complete. By venturing into the shallows of monitoring security, hopefully you will have some direction as to what depths you wish to further navigate.

When we monitor for security, we monitor for three things; Confidentiality, Integrity and Availability. The classic definitions are : [Ref : CISSP Definitions]

Confidentiality
Prevent the intentional or unintentional unauthorized disclosure of a message's contents.

Integrity
Prevent modifications to data by unauthorized personnel or processes, unauthorized modifications to data by authorized personnel or processes, and that the data is internally and externally consistent.

Availability
Ensure reliable and timely access to data or computing resources by the appropriate personnel.

For example, databases containing sensitive data like personnel salaries should be kept confidential, with maintained integrity, while still being available to those persons and processes that are required to create biweekly paychecks.

The depth of your security monitoring depends on the resources you have available to do the analysis. Quantitative security analysis involving estimates of yearly loss per vulnerability takes a long time and is costly. Qualitative analysis (per situation) is easier to perform.[3]

5.2 Strategies

As in the strategies with monitoring performance, determine the baseline of the system. Figure out what the operation norms are. In some respects security is easier since it is more rigid. If we need to ensure that certain files do not change, then we use a tripwire/md5 type solution and compare to our prepared table of hashes.

Some systems come with a predefined set of daily security tests (/etc/security.conf on *BSD systems), the output from these tests can be parsed and a flag raised if need be.

System logs can be analyzed, and patterns alerted on.

Places where changes are made to critical points in your security infrastructure (external access points, LDAP servers, NIS servers, Windows PDC) should be monitored closely as they affect your entire infrastructure if security fails.

Security requires a greater understanding to implement fully. Generally vendors will not share your security model, some unable to grasp that you would care about security at all which results in writing glue code to accomplish your security needs.

5.3 Tools

Sampling of tools for security purposes

nmap – nmap is a ubiquitous tool for scanning networks to see what ports are listening. It is very useful as an early stage in network mapping exercises, and can be used in a script to check for changes to the baseline. (e.g. if a new port suddenly starts listening, you can be notified)

nessus – A security audit tool, nessus performs a range of vulnerability tests against a host via the network.

md5 - MD5 is a hashing tool, producing a small digital fingerprint for any given file. By storing a list of hashes for critical files, you can see when a file has changed unexpectedly.

swatch - 'Simple log watcher', swatch watches syslog files, scans for patterns, and then runs an alert shell script when a match occurs.

/etc/security - *BSD security check shellscript with output that can be parsed and flagged.

portsentry – portsentry binds to a specified set of ports on a host, waits for connection attempts to those ports, and then performs a set of actions (e.g. drop the IP via a null host route, log the connection attempt to syslog which can then be picked up and acted upon by swatch).

logsentry – Similar to swatch, logsentry has more features and understands common log patterns by default. It is also quicker to deploy.

snort – Snort is a packet signature analyzer. It watches packets arriving at a host (or network port if used as a perimeter sniffer), compares those packets to a list of signatures, profiles potential attacks and reports on them.

tcpwrappers – tcpwrappers will monitor connections to daemons (typically started from inetd.conf) and log to syslog. Swatch may be used to alert.

ipfilter/ipchains/ipfw - At a very low level, packets that match specific rules can be logged and/or counted using one of these tools. Counts can usually be displayed with an 'ipfstat' type tool, logs
with an 'ipmon' type tool.

perl - Perl can be the glue code to hold the various other tools together, or to create a new specialised tool.

6. Monitoring setup and configuration

6.1 Centralized monitoring host

Previously, we have mentioned the centralized monitoring host when discussing the tools available for monitoring without explaining what a centralized monitoring host is. The centralized monitoring host , which we will name central, is the system that ties together all of the individual

[3] Security Risk Analysis
http://www.security-risk-analysis.com/introduction.htm

monitoring tools. It should provide the following features :

o Use stable, predictable code. This machine will be left unattended for weeks or even months at a time. You need to be sure that it will still be working as expected when you return
o Implement redundancy features with failover.
o Have configurable alerting of individuals and groups based on different criteria. (time-based, threshold-based)
o Have a variety of standard testing tools for standard services
o Accept input in a standard form from third party software/tools
o Be intelligent about alerting, having the ability to distinguish between a host/service outage, and a network outage.
o Be well written and try to conform to the Unix philosophy of 'small tools to do simple tasks very well'
o Store historical logging of check data for future analysis

In addition, the following features although not required would be optimal:

o Be easy to configure, or have tools to assist in configuration
o Have a nice web display for the command-line challenged (GUIs have their place, this may be one of them)
o Have a nice configurable web display for the company officials to see graphically how the money spent on this monitoring solution is actually saving money.
o A historical reporting function

6.2 The 'uber-monitor'

Quite a few 'uber-monitor' tools have been written, although they are not all currently maintained, or updated. Each has strengths and weaknesses that may determine whether you use one or the other of them in your operating environment. The focus of this paper will be on nagios, as we find it to be the most feature full on average for our needs. Nagios has drawbacks in that it is not very easy to configure, but that problem is actively being worked upon. It is sufficiently lightweight to flex into different designs, yet having a feature set that makes it an asset at the same time.

The principles described should translate to other software tools as well. Determination of tools is a matter of personal choice as stated at the beginning of the paper.

Some alternatives to look at include :
(as mentioned on the Nagios website at http://nagios.sourceforge.net/docs/1_0/about.html#othermonitors)

Angel Network Monitor
Autostatus
Big Brother
HiWAyS
MARS
Mon
Netup (French)
NocMonitor
NodeWatch
Penemo
PIKT
RITW
Scotty/TKined
Spong
Sysmon

6.3.0 An Overview of Nagios

Nagios is primarily a notification tool. By itself it does not do any actual monitoring. Nagios comes with a suite of plugin binaries and scripts to do the monitoring. This means that nagios can focus on the specific task of notification.

Nagios uses basic-auth mechanisms to determine who is allowed to access a given feature. It has several levels of access control within the application, allowing you to define who can view certain info, or modify certain info, as well as other controls. By default owners can only view the service/host checks that they would normally be notified about.

Nagios requires a web server to be installed if you want to run and administer it via the web interface. We favor apache, although any web server that allows CGI should work fine.

Because the basic auth involves sending passwords via the web, it is recommended that you use SSL (https) for the web server and not plain http.

6.3.1 Installation of nagios

Nagios is not yet in pkgsrc. Once this happens the installation becomes considerably less involved.

Install nagios.
1. Create a nagios user and group (both named 'nagios')

2. Install gmake from pkgsrc

```
cd /usr/pkgsrc/devel/gmake
make install
```

3. Download nagios.
http://www.nagios.org/download/

4. Unpack the download file

```
gzcat nagios-1.0b6.tar.gz  | tar vfx -
```

5. Configure and make nagios

```
cd nagios-1.0b6
./configure
gmake all
gmake install  * installs into /usr/local/nagios
gmake install-init  * installs init script into
/etc/rc.d
gmake install-commandmode * installs and sets
permissions on external commands file
gmake install-config  *installs *SAMPLE*
config files into /usr/local/nagios/etc
```

6. Download nagios plugins.
http://www.nagios.org/download/

7. Unpack the download file.

```
gzcat nagiosplug-1.3-beta1.tar.gz | tar vfx -
```

8. Configure and make the plugins.

```
cd nagiosplug-1.3-beta1
./configure
gmake
gmake install  *installs plugins into
/usr/local/nagios/libexec
```

9. Configure your web server as per the nagios documentation. This will vary depending on your existing web server installation. Make sure you have an SSL enabled web server. Mod_ssl works great with Apache, is easy to set up, and has a utility for creating your own certificates.

6.3.2 Editing the Nagios configuration files

You can put all of your nagios configurations into one big file. In practice it is easier to separate different configuration groups into separate files. The main nagios.conf file contains include statements to inherit the other files. This file is the one you specify from the command line when starting nagios.

Although nagios comes with a set of well commented sample files, configuration files are still somewhat bewildering to anyone who has not used the software before. The most common basic actions are shown in the examples below. Following these examples through should give you a minimal working nagios installation monitoring one service on one host.

Set up nagios.cfg (using csh syntax).

1. Make copies of the sample files.

```
cd /usr/local/nagios/etc
foreach i (*-sample)
cp $i ${i:r}.cfg
end
```

2. You do not need to change most of the settings, but you should probably set the values of admin_email and admin_pager in nagios.cfg.

Add a host check.

1. Edit hosts.cfg, by removing everything below the 'generic host template'
2. Add a host check for the host 'foo' by inserting these lines at the end of the file:

```
define host{
        use             generic-host    ; Name of host
template to use

        host_name               foo
        alias                   foo server
        address                 w.x.y.z
        check_command           check-host-alive
        max_check_attempts      10
        notification_interval   120
        notification_period     24x7
        notification_options    d,u,r
        }
```

(NOTE - w.x.y.z is the machines actual IP address, substitue accordingly)

Replace foo with a name of one of your mailservers that runs SMTP.

3. Save the file.

Add a service check.

1. To monitor sendmail/smtp daemon on "foo", edit services.cfg, delete every entry except the one that looks like :

```
# Service definition
define service{
        use                     generic-service    ; Name
of service template to use
        host_name               novell1
        service_description     SMTP
        is_volatile             0
        check_period            24x7
        max_check_attempts      3
        normal_check_interval   3
        retry_check_interval    1
        contact_groups          novell-admins
        notification_interval   120
        notification_period     24x7
        notification_options    w,u,c,r
        check_command           check_smtp
        }
```

Replace host_name novell1 with host_name foo

Notice the two lines contact_groups and check_command, we'll deal with more with those in the next examples.

Configure contacts

If you want nagios to contact you, you need to tell it whom to contact. Let's configure a contact group and the contacts in that group.

1. Edit contacts.cfg.
2. Change the nagios admin definition to have the right email and pager addresses for the person or group that maintains the nagios host.

3. Replace the 'John Doe' example with a real person (probably yourself if you're just starting out).

4. Add any additional contacts in as new records similar to the John Doe example.

5. Save that, and edit contactgroups.cfg

6. Remove every record, except for the 'novell-admins' record (since that is the name listed in our previous service definition, you could of course change it to something else as long as they both match).

7. In the members field, add all of your contacts that you want to be notified for outages of this service, comma separated list.

Configure hostgroups

Hostgroups are a way of grouping functionally similar hosts into groups, for example DNS servers,
mail servers, etc.

1. Edit hostsgroups.cfg

2. Replace the record that reads :

```
# 'novell-servers' host group definition
define hostgroup{
        hostgroup_name  novell-servers
        alias           Novell Servers
        contact_groups  novell-admins
        members         novell1,novell2
        }
```

With :

```
# 'mailserver' host group definition
define hostgroup{
        hostgroup_name  mailservers
        alias           Mail Servers
        contact_groups  novell-admins
        members         foo
        }
```

Edit dependencies

Dependencies are an advanced feature of nagios that allow you to define services and/or hosts as being dependent on other services and/or hosts. This reduces the noise when a given service fails, since monitoring and notifications on the dependent services can be silenced automatically.

At this stage, we will not use this, but the example file comes with some dependencies defined.

1. Edit dependencies.cfg, and comment out every entry by making sure that every line starts with a #.

Edit Escalations

This is another advanced feature that we do not wish to define yet. Basically this is the feature that makes managers very happy, allowing very precise definitions of escalation coverage, at what point different groups get notified about a failure etc... embedding service level agreements in the monitoring.

For now, we shall comment this out as we did with dependencies.

At this point, we can check nagios operation by running /etc/rc.d/nagios reload. You should see something like :

```
gilgamesh# /etc/rc.d/nagios reload
Running configuration check...done
Starting network monitor: nagios
  PID TT STAT    TIME COMMAND
20095 ?? Ss   0:00.02
/usr/local/nagios/bin/nagios -d
/usr/local/nagios/etc/nag
gilgamesh#
```

You can check that things are operating normally without going to the web, simply by looking directly in the status file in /usr/local/nagios/var/status.log (or wherever you have installed in nagios).

Configuring Nagios Remote Plugin Executor

Earlier an alternative to SNMP for monitoring system counters via the network was mentioned. Nrpe is the tool for doing that. The nrpe has two components. One is a small daemon that sits on the monitored host, one is an active client check that runs from central. The daemon has a config file which defines basic IP based access control (which host can get to the daemon and ask it for information, typically only central is defined, perhaps one other IP for redundancy), and then you define a list of strings that you associate with a system command.

Download nrpe.
http://www.nagios.org/download

The build process is in two parts, first compile the check_nrpe client on central.

Download, untar and compile the source on central

```
./configure
make all
cd src
```

Configure nagios commands.cfg to know about check_nrpe

Add the following into commands.cfg :

```
define command {
      command_name    check_remote
      command_line $USER1$/check_nrpe
$ARG1$ -c $ARG2$
}
```

$ARG1$ will be the hostname argument in services.cfg, $ARG2$ will be the name of the check passed to the nrpe daemon on the monitored host.

Compile the nrpe daemon on the remote host (may be a different OS), or copy the compile from central if it's the same OS. I find it easiest to install into /usr/local/nagios on the remote host as well.

/usr/local/nagios/bin/nrpe
/usr/local/nagios/etc/nrpe.cfg
/usr/local/nagios/libexec/check_foo (the plugins you need to use to run the check, either home written in perl/sh, or compiled from nagios plugins)

Define an nrpe.cfg on the remote host, containing the checks you want to execute.

```
# IP address of the monitoring host (hideout)
allowed_hosts=w.x.y.z
#
# list of commands that may be executed locally
#
command[woprdisk]=cd /net/wopr; cd /;
/usr/local/nagios/libexec/check_disk -w 2%
 -c 1% -p /net/wopr
```

(in this case we're using an NFS mount to check how full a NetAPP filesystem has become, it will report a warning at 98% full, and a critical at 99% full. These values were chosen based on the total size of filesystem (> 1TB), typically you would warn earlier on smaller filesystems.)

So say we put an entry like :
command[rootdisk]=/usr/local/nagios/libexec/check_disk -w 10% -c 5% -p /

Startup nrpe on the monitored host, either from inetd, or via an rc script.

Going back to central, define a service check thus in services.cfg :

```
define service {
      host_name             foo
      service_description     foo root disk usage
      check_command
check_remote!foo!rootdisk
      is_volatile                 0
      check_period                24x7
      max_check_attempts          3
      normal_check_interval       3
      retry_check_interval        1
      contact_groups              novell-admins
      notification_interval       120
      notification_period         24x7
      notification_options        w,u,c,r
      }
```

Restart nagios with /etc/rc.d/nagios reload. Now NRPE will check root disk status on foo periodically.

6.3.3 Additional Nagios tools

Nagios Administration Tool (NAGAT) - A web based solution written in PHP for configuring nagios host,service checks etc..

Nagios Service Check Acceptor (NSCA) 2.1 - A two part client/server tool (similar to NRPE) used in the other direction, allowing remote clients to submit asynchronous events (such as security alerts) to a daemon listening on central. The daemon then pushes those events into nagios as a PASSIVE check.

nagios_statd - Perl/Python plugin that lets you check remote host information such as load, users, filesystems etc.

NTray 0.91 - Handy NT app that sits in your system tray and retrieves info from the nagios status file and gives red green lights for you to watch.

Remote Execution Layer (REL)- A layer for providing alternate transport between client and server for NRPE and NCSA. This gets around modifying firewalls to work with nagios. Currently it sends results via email into nagios.

remote_ctl - Perl CGI for easily enabling/disabling service checks remotely using wget or a web browser.

6.3.4 Nagios gotchas

Process space

Nagios can fire off enough checks at one time on the central host that it runs into per-user process limits. Always increase the process limit for the nagios users as you increase your use of the application.

Always do restart

When modifying nagios config files, always use RCS as a matter of course, but never stop and start the daemon, always use /etc/rc.d/nagios restart. The reason for this is the restart option will cause a configtest
against the files before any action is taken. This means that even if you have caused an error in your config files, the running daemon will not go away, and you will get the chance to fix your config files at your leisure without interrupting service.

Use dependencies and parents

Parents are how nagios attempts to model the network and distinguish between network and host/service outages. For every host you define, you should define a parent as the first hop on a traceroute from that host to central. This way, if a network fails and you can't get to a critical server subnet from central, you won't get a number of pages about the hosts, since nagios will know that it's a network outage.

Similarly for dependencies, dependencies actually allow you to monitor more complex systems and reduce the number of 'noise' notifications you get when a failure occurs.

6.4 Monitoring the monitor.

Quis custodiet ipsos custodes - The watched shall watch the watchers

If central fails for whatever reason, you have lost the ability to monitor everything. Unless you are also monitoring central itself, you will never know this as a failure in your monitoring solution as the failure mode is no alerts which is the same as if everything is running smoothly.

A second box should be setup (monitored by central, naturally) whose purpose is to monitor central. It needs to have at least the ability to

inform someone if/when central fails. It should probably complain very loudly when this occurs. Pager notifications to everyone are appropriate in this situation.

Generally the two boxes will not fail at exactly the same time, so you will get some sort of notification about a failure.

6.5 Notifications

You will need two methods of sending notifications. The general day to day method is alerts via emails to email accounts, or pager/sms gateways. The second method should be 'out of band'. Out of band means a method that is not in
any way subject to the same set of failure modes as the original method. If your email server fails, you want to be able to get notifications, but if you rely solely on email to receive your alerts, you will never receive the alert.

Now, you can of course take this requirement to extremes, but generally speaking, it's acceptable to have a separately powered machine, connected
via a phone line that doesn't go through your main company exchange. This machine will be used to send notifications (to a pager usually), via a dialup connection to the pager provider, or to an ISP somewhere.

Again the key here is not to be totally redundant, but to have enough redundancy to be able to see the problems as they occur and be able to react in a timely manner.

7. Monitoring Overlap

As you can see, there is a lot of overlap between the different monitoring categories. Separation of categories is provided to make it easier to understand the different components required in monitoring your environment.

By designing your monitoring tests to complement one another it's possible to make early informed judgments about where problems lie. Try to emulate the sort of questions you would ask in debugging a problem. By understanding what a series of failures occurring at the same time really means, you can jump straight to the underlying problem and fix it more quickly.

8. The physical world

The worst failures often arise outside of the computer hardware or software. Failure of power, UPS, or air conditioning can be catastrophic. A/C failure can go unnoticed for several hours on a weekend, driving up data center temperatures into triple digits creating lumps of metals out of your expensive critical machines.

It is a simple matter to build (or buy) temperature probes which can then report back to the central host. Place several of these around your data center and you then have a handy temperature monitoring system. Graph it with rrdtool over time and you can see if there is a gradual increase and if you have to respecify the A/C coverage. You may also be able to take advantage of the built-in temperature probes in (for example) some Cisco hardware. See references for websites that either sell or have plans for this.

Good quality UPS systems will often have a serial connection which changes state when the UPS operates. This can be monitored, or have an action associated with it (for example, shutting down critical servers gracefully in the time left with power, to save data).

Electronic entry systems generally come with their own software, but if you choose to roll this into your monitoring system it can be done with a little creativity. For example, if you really need to be notified whenever someone enters a particular location for example, or between certain hours. Your imagination is the key.

9.0 Implementation

9.1Case Study I

Divisional monitoring

This system was setup to augment existing monitoring systems in the division, various scripts using OS native tools like ping, and df. It was originally intended as an interesting experiment and cool hobby for the designer, something to play and learn with. Although more of a personal monitoring tool than a production monitoring solution, it did get used by other members of the team. It monitored Irix, and Solaris systems as well as Fore systems network equipment.

Hardware

Single old Pentium 133Mhz box (lethe), running NetBSD1.3, installed with nocol from pkgsrc.

Machine was located on the same subnet and network media as most of the services it was to monitor. Single network interface (10bT).

Relatively new to NetBSD at the time, I found it a much better operating system to work with, than the other operating systems in use at the organization. Irix was going through a period of flux in its OS development, 32bit versus 64bit, n32 versus o32. Compiling was a nightmare sometimes, especially with Open Source software. Solaris was clunky. Nocol was chosen, as it best met needs at the time from the packages that were researched online, and from pkgsrc.

Implementation

It took a few weeks to configure, as this was a first attempt at this sort of service. Approximately 40 Unix servers were monitored with ‘rpcpingmon', and 15 managed switches and routers were monitered with ippingmon. Critical servers’ performance statistics were monitored more closely with 'hostmon'. Some experimentation with syslogmon occured, but at the time, a separate system using swatch by itself was in use.

Notifications were via email and pager, and also via a console ('netconsole') running on the network administrator's workstation.

After setup, I examined the system every time new systems were installed into the environment. I also reevaluated the configuration every few months to fine tune the system.

The system scaled reasonably well, although some slowdown was noted on lethe as more checks were added. There was no provision in the system to understand network outages, and no redundant notification paths. Given the environment this was acceptable however, since most of the admins were frequently online and watching stuff anyway.

9.2 Case Study II

Enterprise wide monitoring setup

This system was setup from scratch in a place with no monitoring. It was intended to completely monitor all critical services within the enterprise. At this time it is still evolving

Hardware

Three desktop PCs (Pentium II-400, 128MB RAM, 4 or 9GB hard drives), each with a vanilla NetBSD installation (1.5.x).

PC1 (hideout) - The main nagios host, some sample configuration files shown in Appendix B.

PC2 (pageboy) - The phone dialer. Disconnected completely from the network for security reasons (the dialing process will bring up an IP stack), and connected via serial cables to PC1 and PC3. Communications via UUCP. This is the host that will send out pages via an external service. The connected analogue phone line doesn't go through the main building phone switch.

PC3 (hwatch) - Hideout's watcher. This is the second monitor host whose only purpose is to monitor PC1 and scream if it goes down. Always sends pages via pageboy. Arguably this function could be rolled into pageboy.

With many years of NetBSD experience now, NetBSD was the obvious choice because it’s great and has never failed at any of the tasks I’ve attempted with it. I’m very comfortable with the OS. Nagios was chosen, as it seems to be the best of the monitoring packages available currently. that.

Implementation

Setting up the initial install of nagios took an afternoon. The longest part of configuring nagios, was gathering information from other groups to make it useful to them. The basic system monitoring was up within hours, and configured more completely in about 3 weeks. The first couple of days, I configured nagios to monitor the infrastructure (DNS, mail, NIS, LDAP, etc..), the rest of the time was adding in hosts that others needed, using nrpe to gather system specific statistics.

The system currently monitors 72 hosts and 126 services. No noticeable performance impact has been seen on hideout, notifications (based on time tuning) are generally immediate, and problems are noted and worked upon before anyone in the user community sees them

Since setup, I regularly check the system weekly to make changes. It is gradually getting into people’s awareness as something to consider as part of a deployment of new services.

I've written specialized nagios plugin client code, to check our high uptime web content.

It has assisted our organization by allowing us to respond to failures before they become catastrophic, or impacting business profits. Our group has a better reputation, with more personnel having faith in the IT department's ability to prevent and handle emergencies.

Setting up a similar system

Generally there are very few limitations unless you're really deploying the system worldwide, at which point your better option is to scale hierarchically and have slave nodes reporting back to a master reporting station. Current desktops have more than enough resources in every area to perform as a central monitoring host. The current enterprise setup is not suffering under its current load. Figuring out the ideal system really depends on how far you want to scale.

If you need to buy a system, a mid range rack mount or desktop will be fine. A laptop will work in a pinch. High disk I/O isn't absolutely necessary (although it might make response time better when you need to maintain the system).

My ideal setup would probably be an older model athlon on a stable chipset board with 512MB ram and 18GB mirrored disk. That would be ample for a standalone station.

If you are at the point of needing slave nodes, then use a similar setup, using a current model athlon, bigger disks with hardware raid, and gigE connections for the master node.

Future work

When the world of the dot-coms has money once again it will be nice to build a monitoring solution with more powerful machines, along with the usual hardware redundancy features. In choosing the hardware (they are currently compaq PCs...) we will be more easily able to raidframe disk mirroring etc...

Conclusions

We defined monitoring as a sampling of some sort of content, systematically tracking the state of that content, and warning the appropriate parties when needed. By investing time and resources on preparing a monitoring solution at the outset of enterprise architecture, catastrophe will be adverted when AC in your data center fails, the company's website becomes unreachable or a distraught recently fired HR employee attempts to trash the systems. A monitoring solution does not replace the need for redundancy in your systems, or having reliable backups. As system environments become more complex, monitoring becomes more important. With the lag in resources for increasing manpower versus the need for more systems to handle load, a good monitoring solution is the only way to keep on top of system performance. Running NetBSD, one of the most stable and secure operating systems available, with the very configurable nagios, you will have built a monitoring solution that will withstand most system and network administrator's nightmares. Instead of Mr. Smiley calling you up questioning the cost benefits of your setup, and monetary losses of downtime, you will be prepared with information to backup decisions on buying system/network resources, and be prepared when emergency strikes to minimize downtime.

The intent of this paper was to impart design skills to help you enhance your own monitoring solutions, and get you started with a basic monitoring framework if monitoring is a new topic for you. The authors are happy to field questions via email, and contract work is always welcome.

References

Online copies of this paper, sample perl tools source code and sample Nagios configuration files may be obtained from http://www.deorth.org/papers/monitoring

Mr Smiley http://www.userfriendly.org

Obtaining pkgsrc http://www.netbsd.org/Documentation/software/packages.html

Fping
cd /usr/pkgsrc/net/fping && make install.
Or, sources can be obtained from
ftp://ftp.uu.net/usenet/comp.sources.unix/volume26/fping/

SNIPS http://www.netplex-tech.com/software/snips/
Nagios and associated plugins and contributions http://www.nagios.org
TKined (and in kpgsrc) http://wwwhome.cs.utwente.nl/~schoenw/scotty/
Lynx (and in kpgsrc) http://lynx.browser.org/
Wget (and in pkgsrc) http://www.gnu.org/software/wget/wget.html
Big brother http://www.bb4.com
Big sister http://bigsister.graeff.com/
MRTG (and in kpgsrc) http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool
Cricket http://cricket.sourceforge.net/
Flowscan http://net.doit.wisc.edu/~plonka/FlowScan/
Smokeping http://people.ee.ethz.ch/~oetiker/webtools/smokeping/
Other front ends to rrdtool http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/rrdworld/
Nmap http://www.nmap.org/nmap/index.html
Nessus http://www.nessus.org
MD5 Part of the NetBSD base OS, src for compiling on other operating systems is available from
ftp://ftp.cert.dfn.de/pub/tools/crypt/md5/01-README
Swatch http://oit.ucsb.edu/~eta/swatch/
Portsentry http://www.psionic.com/abacus/portsentry/
Logsentry http://www.psionic.com/products/logsentry.html
Snort http://www.snort.org/
TCPwrappers ftp://ftp.porcupine.org/pub/security/index.html

Acknowledgements

We'd like to thank the BSDCon committee for giving us the opportunity to share our thoughts with the BSD community.

Alan would specifically like to thank his places of employment Inktomi and Dreamworks for providing the work environment to experiment and explore innovative ways of providing better system performance through monitoring. Also, he would like to thank David Brownlee for introducing him to the one OS, NetBSD, and the many people in the NetBSD community who have always been helpful and instructive through the years.