7

Before Things Break—Building a Baseline

SOMETIMES WHEN YOU TALK TO A SEASONED SYSTEM or network administrator, he'll tell you that he knows that something is wrong when things don't feel right. This isn't an admission of paranormal powers; it's just a shorthand method for explaining that these experts know how their system or network is supposed to behave and that it isn't acting like that now. These administrators have created a baseline for their environment. Not all of them have done it formally, but the ones who have will have gained significant added benefits.

In this chapter, we'll cover what a baseline can do for you, what it really is, how to create one, how to keep it up-to-date, and how your baseline affects your network monitoring plan (and vice versa).

Why Baselines Matter

Baselines give you two primary benefits. First, they let you know how your network behaves under normal conditions—this lets you see where something is going wrong while you're troubleshooting a problem. Second, they let you see how your network's behavior is changing over time—this will help you maintain a healthy network through careful expansion. Both of these provide you with some real value.

A Baseline, Your Troubleshooting Friend

While you're troubleshooting, you'll use your baseline in two ways. The most obvious use is that you'll be able to tell when things are behaving outside the norm for your network. It is less obvious, but just as important, that you can use the baseline as part of your diagnostic tool kit. When taken together, these two benefits make your troubleshooting life much easier.

When you're aware of your network's normal behavior, changes become obvious. If Wednesday afternoons feature low bandwidth utilization but a high load on one of your servers, you won't be concerned if that server is slow to respond while the others are okay. All of the servers being slow to respond would send up a warning flag right away, though. This awareness cannot replace monitoring your network, but it does serve as a good adjunct to it.

When in the midst of troubleshooting, having your baseline information accessible can make a world of difference. For example, if you know that hosts cherry and berry aren't typically big users of bandwidth, but they seem to be the culprits in your current network overload, you've got a head start on identifying the problem. If you further know that one or both of them were just upgraded, you have an even better map. (Of course, it could always be the users flexing their Quake muscles…)

Having baseline information about your network will help you spot problems earlier and solve them faster. It may take a bit of work to get it started, but the effort expended now will pay off in weekend and evenings free later!

Watching Your Network for Fun and Prophet

Beyond helping you with your troubleshooting, a network baseline will help you avoid problems that you would otherwise walk into. Watching your network's behavior change will help you see where you need to make changes. Is traffic building up on a LAN segment? Maybe you'll need to add another switch. Do you have a server acting as a bottleneck? It's probably time to upgrade.

As your base of users grows and their needs change, your network will need to change as well. When you watch your network and can foresee needed changes in topology or equipment, you'll save yourself time and money. This is one of the best reasons to not only develop a baseline, but also to actively monitor against it and keep it up-to-date.

What Is a Baseline?

Several things make up a baseline, but at its heart, a baseline is merely a snapshot of your network the way it normally acts. The least effective form of a baseline is the "sixth sense" that you develop when you've been around something for a while. It seems to work because you notice aberrations subconsciously because you're used to the way things ought to be. Better baselines will be less informal and may include the following components:

- Network traces
- Summarized network utilization data
- Logs of work done on the network
- Maps of the network
- Records of equipment on the network and related configuration data

Each of these will be discussed in this section.

Network Traces

In Chapter 10, "Monitoring Tools," we discuss the ethereal network analyzer. This tool's capability to save capture files (or traces) enables you to maintain a history of your network. If the only traces you have saved represent your troubleshooting efforts, you won't have a very good picture of your network.

You also need to be aware that a lot of things will influence the contents of the traces you collect. Weekend vs. weekday; Monday or Friday vs. the rest of the week; and time of day are all examples of the kinds of factors that will affect your data. Running ethereal (or some other analyzer) at least three times a day, every day, and saving the capture file will give you a much clearer idea of how things normally work.

Utilization Data

Several tools can give you a quick look at your network's behavior: netstat, traceroute, ping, and even the contents of your system logs are all good sources of information.

The netstat tool can show you several important bits of information. Running it with the `-M`, `-i`, and `-a` switches are especially helpful. I typically add the `-n` switch to netstat as well. This switch turns off name resolution, which is a real boon if DNS is broken or IP addresses don't resolve back to names properly. The `-i` switch gives you interface specific information:

```
[pate@cherry sgml]$ netstat -i
Kernel Interface table
Iface   MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   0        0      0      0      0       39      0      0      0 BRU
lo     3924   0       36      0      0      0       36      0      0      0 LRU
[pate@cherry sgml]$
```

The -M switch gives information pertaining to masqueraded connections:

```
[pate@router pate]$ netstat -Mn
IP masquerading entries
prot   expire source               destination          ports
tcp  59:59.96 192.168.1.10         64.28.67.48          1028 -> 80 (61002)
tcp  58:43.75 192.168.1.10         206.66.240.72        622 -> 22 (61001)
udp  16:37.72 192.168.1.10         209.244.0.3          1025 -> 53 (61000)
[pate@router pate]$
```

The -a switch gives connection-oriented output (this output has been abbreviated):

```
[pate@cherry pate]$ netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:111             0.0.0.0:*
raw        0      0 0.0.0.0:1               0.0.0.0:*               7
raw        0      0 0.0.0.0:6               0.0.0.0:*               7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  1      [ ]         STREAM     CONNECTED     1332   /tmp/.X11-unix/X0
unix  1      [ ]         STREAM     CONNECTED     1330   /tmp/.X11-unix/X0
unix  0      [ ]         DGRAM                    440
[pate@cherry pate]$
```

The traceroute tool is especially important for servers that handle connections from disparate parts of the Internet. Setting up several traceroutes to different remote hosts can give you an indication of remote users connection speeds to your server.

The ping tool can help you watch the performance of a local or remote network in much the same way that traceroute does. It does not give as much detail, but it requires less overhead.

When users connect to services on your hosts, they leave a trail through your log files. If you use a central logging host and a log reader to grab important entries, you can build a history of how often services are used and when they are most heavily utilized.

Work/Problem Logs

You will likely find yourself touching a lot of the equipment on your network, so it is important that you keep good records of what you do. Even seemingly blind trails in troubleshooting may lead you to discover information about your network. In addition, you will find that your documentation will be an invaluable aid the next time you need to troubleshoot a similar problem.

Some people like to carry around a paper notebook to keep their records in; others prefer to keep things online. Both camps have good points, many related to information access. If you keep everything in a notebook but don't have it handy, it does you no good. Similarly, if everything is online and the network is down, you're in bad shape.

My preference is to keep things online, but in a cvs repository. Then you can keep it on a central server or two while also keeping a copy on your laptop, PC, or palmtop. If you like, you can even grab printouts. A nice benefit to this is that several people can make updates to documentation and then commit their changes back to the cvs repository when they've finished.

I won't get into the Web vs. flatfile vs. database vs. XML vs. whatever conflict. They all have benefits. Choose the right option for your organization, and stick to it. The important bit is that you have the data, right?

Network Maps

A roundly ignored set of baseline information is the network map. If you have more than two systems in your network and don't have a map, set down this book for 20 minutes and sketch something out. It doesn't have to be pretty, just reasonably accurate. Are you back? Good. Now that you have a map showing what is where, we can get back to work.

Most people want to deal with two kinds of maps. The first is a topological/physical map, which shows what equipment is where and how it is connected. The second is a logical map. This shows what services are provided and what user communities are supported by which servers. If you can combine these two maps, so much the better; color coding, numeric coding, and outlined boxes are all mechanisms that can help with this. A sample map is shown in Figure 7.1.

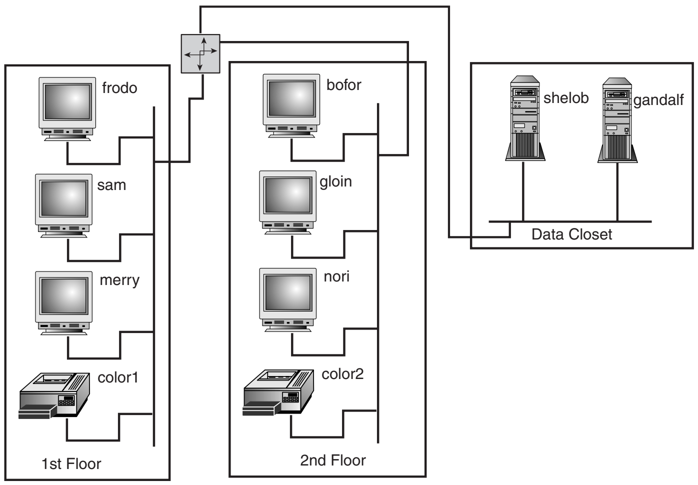

Figure 7.1 A sample network map.

Like the information discussed in the preceding "Work/Problem Logs" section, I recommend that you keep your maps online and in a couple of places. (cvs can be a good solution here as well.) Nicely done maps also look good on your wall, not to mention that this is a convenient place to find them when a problem breaks out and you need to start troubleshooting.

Equipment Records

You should also have accurate records of the hardware and software in your network. At a minimum, you should have a hardware listing of each box on the network, a list of system and application levels (showing currently installed versions and patches), and configurations of the same. If you keep this in cvs, you'll also have a nice mechanism for looking at your history.[1]

If you decide to keep these records, it is vital that they be kept up-to-date. Every time you make a change, you should edit the appropriate file and commit it to cvs. If you fall behind, you'll miss something, and then you'll really be stuck.

How to Create a Baseline

Okay, I've talked a lot about what you should have, and I've created a daunting list of stuff that you ought to do. How are you going to get it done? Never fear, most of this can be done in small chunks. In addition, you'll find that much of the work can be scripted and then run from cron.

In this section, I'll cover the following:

- Deciding what to do first
- Following naming convention
- Keeping logs and records
- Making maps
- Using cron to do your dirty work for you

What Do I Do Now?

To me, your baseline is pretty useless without a map and inventory. With this in mind, I recommend that you do these second—yes, second. The first thing you'll want to do is to decide how and where you're going to keep all your data, and then set it up. (While you're at it, you might as well start your own logbook. You'll be touching a lot of things while you make your map and inventory, so you'll get a chance to start making entries.)

When you've got a repository for all your data set up (you did use cvs, right?), and when your maps and inventory are in it, you're ready to start on the next step. This is a good time to lay out a plan for what you want to watch and how often you want to check it. Some things should run constantly—log file watchers are a great example. Others, such as ethereal, traceroute, and netstat, will need to be run periodically.

Which of these you implement first depends a lot on your situation. Do you have any outstanding problems? If so, start building your baseline around your troubleshooting. Are you especially interested in watching certain areas? Start watching them. I think that it works well to start with something you're familiar with. You'll gain confidence in your plan and in your scripts with each success.

Another idea that bears mention is testing your baselining tools on small parts of your network. When you've got things working on a subsection, it is much simpler to extend them across your entire network.

Good Naming Conventions—Keeping It All Straight

Assuming that you're not throwing all your baseline data into a database, you're going to want a way to keep track of it. Even if you track only a couple of things, you'll soon have a huge collection of files. Without a way to track it all, it will be a mess of epic proportions. Your naming convention will go a long way toward helping here.

A large-grained approach is to separate all your files into monthly directories. From there, you can either divide like files into subdirectories or divide your files into daily subdirectories. Regardless, you'll want to name each file clearly showing what it contains, the date and time it was created, and which host it was created on (for example, ethereal-20001025-0900-mango_eth0.cap for an ethereal capture created on eth0 of mango at 9 A.M. on October 25, 2000). If you use different switches on a command, you might include that in the filename (for example, netstat-an-20001025-0900-mango, netstat-in-20001025-0900-mango, and netstat-Mn-20001025-mango).

Follow Naming Conventions Closely

Whatever naming convention you decide on, follow it religiously! You will want to be able to access the files from scripts, and a common naming scheme will make this far easier. If you ever decide that you do need to change your naming convention, having a regular system to change from will also make your life more pleasant.

Getting a Record Started

Starting your host-by-host record may seem like the hardest task of those outlined, but it can actually be pretty simple. Before I dive into hands-on advice, I'd like to weigh in on flatfiles vs. a database for system records. I think it's important to keep a local copy of this record on each host. To me, the best way to do this is to update it on the local host and to commit your changes to cvs after each change. This way, even if the box is isolated from the rest of the network, you have an accurate copy to work from.

Deciding what and how much to keep may seem like a difficult task, but (with the help of your system tools) it's pretty easy.

I recommend that you keep a list of installed software; if you use rpm to manage your system, this is not difficult. You can use the following command:

```
[root@cherry cherryconf]# rpm -qa |sort >rpm-qa
[root@cherry cherryconf]#
```

This will create a list of all installed packages in the following format:

```
ElectricFence-2.1-3
GConf-0.5-0_helix_2
GConf-devel-0.5-0_helix_2
Gtk-Perl-0.7003-0_helix_2
ImageMagick-4.2.9-3
MAKEDEV-2.5.2-1
```

Because the records are sorted, they are suitable for comparison with tools such as comm.

Using rpm can cause a bit of additional work for you because not all packages come in rpm packages. If you choose to use rpomm to simplify your record keeping and maintenance, you'll want to learn how to build your own rpms. This is not a difficult task, but it will require a bit of ramping up. The basic procedure is to move a tarball of the application you are planning to build into the /usr/src/redhat/SOURCES directory (it should be named foo-version.tgz). Then create a spec file in /usr/src/redhat/SPECS. Here's a sample spec file:

```
Summary: net-fu - a network foomigator
Name: net-fu
Version: 0.1
Release: 1
Copyright: GPL
Group: Applications/Terminal
Source: http://netfu.org/source/net-fu-0.1.tgz
Distribution: RedHat
Vendor: N/A
Packager: Pat Eyler

%description

%prep

%setup -q

%build
configure —prefix=/usr/local
make
```

```
%install
make install

%files
/usr/local/man/man1/net-fu
/usr/local/bin/net-fu

%clean
cd ..
rm -rf net-fu-0.1

%changelog
* Thu Dec 7 2000 Pat Eyler
- packaged as an example of an rpm
```

The %files Section

You'll want to pay special attention to the %files section. It controls which files are installed and managed by rpm. If the file isn't listed there, it won't be installed even if your `make install` would normally install it.

You'll also want to track the interface configuration and any access control configuration you have. Some of this information can be extracted automatically from the system, such as using `ifconfig -a` to collect interface information. Other bits of it must be read from config files such as /etc/hosts.allow or /etc/hosts.deny.

In addition to configuration information, you'll want to keep track of hardware installed on you server. This is a little bit harder to do, but it is well worth the effort. In this case, keeping a system's record on that system in addition to a central location is a good idea, but you might want to think about keeping it in a format that is easy to parse so that you can quickly find all the systems with similar hardware if you find yourself in the middle of a component recall (or similar situation).

Map Making

Map making can be a bit of an art, but anyone can make a serviceable map. Linux and GNOME even provide some great tools to help. One of my favorites is dia, a GNOME diagramming tool written by Alexander Larsson, James Henstridge, and a host of contributors. dia will give you a canvas to work on and icons representing systems on your network.

A simple map might look like the one in Figure 7.2.

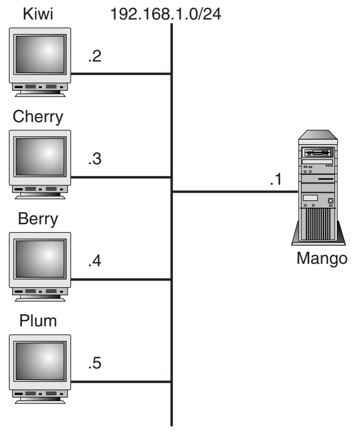

Figure 7.2 A simple network map.

A map that is this simple is going to rely on external documentation to keep track of many things, but it does show IP addresses and the mask for the network.

A more descriptive map is shown in Figure 7.3.

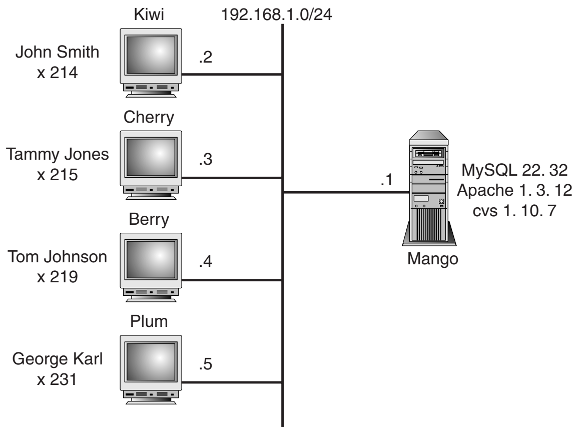

Figure 7.3 A more complete map.

This map includes contact information for workstations and information about which services are provided by a server. You could further improve the map with color coding to show additional information such as user department or project. Notes showing the physical location of each host or actual network connections to hubs or switches would also make your maps more useful.

Using cron

When you've started gathering this information, you'll want to automate it. Automating information gathering helps you in two ways: First, it enables you to collect the information at odd intervals (especially at times when you're not there). Second, it enables you to ensure that each item is gathered the same way every time.

For example, a convenient method of running ethereal every hour at 15 past the hour might be to run the following script from cron:

```
#!/bin/bash

export HOST=`hostname`
export DATE=`date +%y%m%d%I`

export FILE="${HOST}.${DATE}

tethereal -c 5000 -t r -w /var/log/ethereal/${FILE}

cd /var/log/ethereal

export CVS_RSH=ssh
cvs add ${FILE}
cvs commit -m "latest capture" ${FILE}
```

To run this script from cron, you will need to add a line like the following to the root crontab[2] (remember that only root can run ethereal/tethereal unless you allow other users to open an Ethernet interface in promiscuous mode):

```
15 * * * * /usr/local/bin/run_tethereal > /dev/null 2&>1
```

When you've put the whole thing into cron, including the cvs updates, you should write a script that ensures that all the changes were made by checking your cvs log files. There's nothing worse than having a nice data collection system and not noticing that some of the data isn't being collected. A cron-based script to do this is left as an exercise for the reader.

How to Keep Your Baseline Up-to-Date

There are two sides to keeping your records current. The first aspect is that you need to have an entry for every test each period; cron should take care of this for you. The second is that you need to add each new system into your baseline as it is added to your network. This one needs to be done by hand (well, sort of).

One way to help ensure that things get added is to make adding information as painless as possible. Again, we can turn to the system administrator's standby: scripting. If you write a script to add a new host and let that script take care of creating and adding the appropriate entries in the appropriate places, you'll find that your life seems a lot less complicated. You won't forget to do that one thing that always seems to be forgotten. You may even be able to hand the initial data entry off to the user of the equipment.

The `adduser` command is a good example of this. A better solution for host information might be to provide a script that the user can run when installing the system that puts the updates into a central repository. The needed additions and corrections to local configuration files can then be made by collecting data from this central repository.

If all my warnings in the last section weren't enough to convince you to use cvs and a naming standard, this is my last swing at it. Having these in place will make automatic maintenance and creation of the per-host files a lot simpler. Simpler means fewer mistakes and a greater chance that things will get done. It's a good idea. Do it.

Where Monitoring Fits into All of This

Okay, so where does monitoring fit into all of this? A lot of what I've talked about is monitoring, after all. Is this all you need to do? Not really.

Although the monitoring described previously helps you watch the general direction of your network and the ways it compares to the norm at any given time, it really represents only an occasional snapshot of the way things are. True monitoring is a near real-time view of reality. The two types of monitoring are complementary.

Monitoring should include frequent checks of systems and applications that you want to keep available to your users. You also want to ensure that you find out about failures through some alarm mechanism (pagers, email, or the like). Your monitoring plan will likely overlap your baseline by a bit. If this is the case, you should look into ways to combine the two functions.

The mon tool, which is discussed in Chapter 10, is a wonderful utility for watching your network. It enables you to send alarms to various users based on the results of your tests. Although mon is designed as a monitoring tool, extending it to write data into a record for your review is pretty trivial.

You might find that your baseline indicates that you need to adjust your alerting thresholds at times. This is really just another benefit of keeping a baseline. It helps keep your view of reality current. You might find that the reverse is true—something in your monitoring might suggest that you should add a test to your baseline. Again, this is just another benefit of really watching your network.

Endnotes

1. cvs is just good stuff. If you don't have it, you should go get it. And, no, I'm not a maintainer or a reseller, or otherwise involved with it.
2. To add a line to the crontab, use the `crontab -e` command.