USENIX®

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

ISBN-13: 978-1-931971-70-6

90000

9 781931 971706

20 Daniel V. Klein and John Sellens Running the Numbers: System, Network, and Environment Monitoring

20 | *Short Topics in* System Administration

Jane-Ellen Long, Series Editor

Running the Numbers: System, Network, and Environment Monitoring

Daniel V. Klein and John Sellens

USENIX®

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

20 *Short Topics in*
System Administration

Jane-Ellen Long, Series Editor

Running the Numbers: System, Network, and Environment Monitoring

Daniel V. Klein and John Sellens

Published by the USENIX Association
2009

ISBN 978-1-931971-70-6

To purchase additional copies, see http://www.sage.org/pubs/short_topics.html.

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA USA 94710

http://www.usenix.org/

First Printing 2009

Contents

Preface

To explain our motivation for writing this booklet, we give you Dan's story:

> It started innocently enough. I ran a Web server, and it was in my third floor office. The office got hot, the machine got hot, so one day I decided to install a new chassis cooling fan, which meant I needed to reboot my server. The whole process took about 30 minutes. At the end of it, the machine would not reboot. It turned out that the disk had been running far in excess of the rated temperatures and the drive electronics had developed problems. Annoyingly, they worked fine for the months of overtemperature conditions (with brief reboots for operating systems upgrades), but when the machine was shut down for 30 minutes, the electronics cooled off more, shrank, and . . . cracked. No disk meant no Web server—and no income!
>
> Everyone asked, "Didn't you have backups?" Well, of course I did—daily, weekly, and monthly! Except a few months before, I had changed something "unimportant" in my backup scripts[1] and never really tested them again (I just assumed that they'd work). And it turned out that I had a collection of nearly empty backup tapes. Fortunately, the disk platters were undamaged, and for a mere $6,000 to the Drive Savers disk recovery service I was able to recover all the data. I was back up and running in a little over week. So much for five sigma reliability.
>
> So I bought a window air conditioner for the summer months, and whenever it got hot, I'd turn it on. Since I was down $12,000 in lost income and disk recovery costs, I was motivated. But I also traveled a lot for work, and I didn't want to leave the A/C on for weeks at a time if I didn't need to.
>
> I needed a way to remotely monitor the temperature of my office and, more importantly, the temperature of my servers. Thus was the innocent beginning for what has become a minor obsession.
>
> I found an inexpensive $25 kit that would allow me (through my computer's serial line) to measure temperature in up to four locations. No special software was required—you could just connect to the serial line and look at the current temperatures, so I could see the server temperature anytime I was connected to the network. But as long as I was looking at temperatures, why not log a history of data? So I wrote logging software that would record the temperatures, and then I added extra cabling and sensors. Now I was keeping track not only of the server temperature but also of the ambient room temperature, the outside temperature, and the temperature in the basement.

1. I had broken the order and types of the backups. Instead of writing three disks to the non-rewinding tape and writing the last one to the rewinding tape device, I wrote two disks to tape and then rewound, writing the third and fourth disk to the rewinding tape, overwriting the first three backups with the last backup. And that disk was last because it wasn't very important!

As long as I was at it, I wrote graphing software that would allow me to look at the day's temperature, the week's, or the month's. Then I added moving graphs, so that I could look at specific days in the past. I could tell you when the furnace came on and how long it ran. I could show you how the temperature dropped when it rained on a summer day. I could even show you the blips in the basement temperature that corresponded to the cycling of the dehumidifier. Most importantly, I could ask my housemate to turn on the A/C if the office got too hot. I was hooked.

I built a second system out of an old Novell "pizza box" system (the computer, with disk, was about one foot square and only three inches high) and installed that at a friend's summer camp at Lake George, NY. I built submersible sensors into the test tubes I got when I donated blood, and I sealed them with marine goop. The sensors measured water temperature at the surface and about 15 feet deep, as well as the inside and outside temperatures.

The problem was, my little monitoring kit would only support four temperature sensors and my software would only support a single data collection unit. So I was maxed out both at my house and at the camp. But where there is a will, there is a way. I researched other systems and discovered that there was a wide variety of data collection systems, and a whole family of sensors (including temperature, humidity, wind, rain, barometric pressure, light, and switch sensors).

I wrote one manufacturer that I had this neat monitoring package and would like to expand it. If they'd send me a sample of their hardware, I'd support it and let them have my software. To my amazement, they said yes.

But when they actually *sent* me $200 worth of hardware, I got to work. Because the new device was Ethernet-based (and not serial), I had to completely revise my software design. While I was at it, I revamped the graphing software to be more usable and added support for additional types of sensors. I started looking at more data—and the more you look at, the more you learn, and the more you realize you need to look at even more data.

I wrote to another company. They sent me another $200 sample system. I added support for wetness sensors, rain gauges, anemometers, and wind direction. I revamped the graphing software to handle radial graphs in addition to linear ones. I got braver and wrote to a third and fourth company, and they sent me $300 and $450 samples. I added support for switch sensors, barometric pressure sensors. This was getting scary! People started writing to *me*, asking if I'd add support for their hardware.

I got listed on HackADay.[2] I learned about more data collectors, and now I support a large number of them.

But addictions are hard to satisfy. I now look at my hot water usage with temperature sensors on the pipes. I acquired a power monitoring system (another donation in exchange for software support) and can tell you how much power each circuit in my house is drawing. I have planned a click sensor on my gas meter to measure gas con-

2. http://www.hackaday.com/.

sumption. I have door sensors on the garage and look at temperature and humidity throughout my house.

"Why?" you may ask. Well, other than a hobby gone wild, other than "because I can," there is a very good reason: money!

I bought my turn-of-the-century house in 1980. Since then, I have added insulation everywhere I could. It cost a little and has saved a lot. My furnace was 60 years old, so when I replaced it, I installed a 98.5% efficient furnace. It cost more, but paid for itself in five years. As a luxury, I replaced the window A/C with whole-house air, but I only use that when it is beastly hot. But now that I am monitoring things, I can save even more money.

Running whole-house A/C is expensive, but I discovered that by raising the temperature on the thermostat by one degree, I was able to reduce A/C use by 75% (I can show you the graphs where I compare plenum temperatures to the outside air temperature and humidity). The basement humidifier is essential, but it just has a dial on the front labeled "low" to "high." By monitoring the basement humidity, I determined where I could reasonably set the dial to prevent mold and mildew, yet keep the operating costs down. I've moved my office from the third to the second floor, but the third floor sensors tell me when I have forgotten to close a window on a cold day. And the electrical and gas monitoring help me optimize my energy consumption while still staying comfortable.

But I'm not done yet—a true compulsive never is! I'm going to combine the temperature sensors with dampers in my ductwork, so I can better regulate the temperature in the whole house (right now I manually tweak things, but a scientific, computerized approach will be much better). Is the garden getting dry? They make soil sensors! How much lightning was there during that storm last night? They make lightning sensors! Hey, Alvin and Parker (my cats), are you hungry or thirsty? Why not monitor the water and food levels in the cats' bowls?

That is where this booklet comes from: our personal "obsession" with seeing how things work—because if you watch your systems, you can watch the effect your changes make, and then you can make things work better. And if you watch what effect external changes have on your operating environment, you can prevent problems. We do it "because we can," of course, but more because it makes a difference in terms of efficiency, cost savings, and disaster prevention.

Thanks and gratitude are due Jane-Ellen Long for her encouragement and gentle prodding, through more than a few missed deadlines.

Big thank yous are also due to Tom Limoncelli and to Mark Burgess, and to Michel Matteau and Chay Wesley.

The quality of the finished product is a result of many contributors, named or not; any remaining faults are ours alone.

1. Introduction

This booklet is about one aspect of system and network monitoring: collecting, storing, reviewing, and acting on numbers (or, more formally, data points) that tell us something relevant about our environment. We are concentrating on instrumenting our systems and networks rather than on examining or monitoring the services we provide. We focus on monitoring our environment and less on controlling the equipment that controls it. But with any monitoring process comes the ability to influence the thing being monitored, either automatically or through conscious action on the part of the person(s) doing the monitoring. As Lord Kelvin[1] said, "If you cannot measure it, you cannot improve it," and we seek to improve our processes by monitoring what is going on in them.

1.1 What Is Monitoring?

When we talk about monitoring, we often define it as "periodic sampling of system, device, application, environmental, or network status information, recorded and/or dispatched for analysis and/or correction." Or, more succinctly, "information about systems and networks that is collected, analyzed, and acted upon."[2]

System, network, and environmental monitoring is an important part of system and network administration because it:

- ❖ Contributes to reliability and availability.
- ❖ Helps gets problems identified (and fixed) faster.
- ❖ Contributes to a better overall quality of service.
- ❖ Contributes to a more enjoyable and effective working life for system administrators.

Limoncelli/Hogan/Chalup [Limoncelli07] may have said it best when they wrote, "A service is not complete and cannot properly be called a service unless it is being monitored." If there is no monitoring then you're just running software.

1.2 Why Monitor?

We'll claim that monitoring is done for three primary purposes:

Exceptions and Anomalies

- ❖ To identify and report unusual conditions, failures, possible problems that may need attention.

1. http://en.wikipedia.org/wiki/William_Thomson,_1st_Baron_Kelvin.

2. Which also includes inaction: a decision not to act is still a decision.

Examples: Is the Web server serving up Web pages? Is the disk full? Is the data center on fire?

Trends

- ❖ To collect numeric data that characterizes some aspect of a system or network over time.

  Examples: Network bandwidth utilization, CPU utilization, Web site hit counts, number of bytes used on a file system, the temperature in the data center.[3]

History

- ❖ For historical analysis of time-stamped data, primarily for record-keeping and trouble-shooting.

  Examples: Billing, tracking who was doing what and when, or responding to discreet (or indiscreet) inquiries from law-enforcement officials.

In this booklet we will concentrate primarily on data collection and monitoring for "trends," with some identification of exceptions. We are primarily looking at the collection of information that can be quantified numerically, and how we can visualize and analyze the data we have collected over time. Some of the numbers we collect may be simple binary state information (e.g., whether a door is open or closed, a light is on or off, there is water on the floor or not), but in general we're not going to be looking at methods for checking the state of services (e.g., whether the mail server's SMTP port is communicating properly, or whether the network connection is up or down). For trend monitoring, we're more interested in the *volume* of traffic for a port or service, but if you have a ready means of determining application state[4] or port status, that state information can often be supplied as a binary input to the monitoring software we are describing (see section 2.4.9). If the numbers we collect have values that are too high or too low, we may have an indication of an "exceptional" condition that we will want to report and/or act on.

Unfortunately, monitoring is not a task that has the benefit of immediate time savings. Like most tasks, setting up a robust monitoring system will take time and will likely require that you learn how to use a few different tools. Monitoring may also never save you time on the back end either—if you adhere to a rigorous maintenance schedule and have perfect firewalls, all your monitoring will ever show you is that things are ok. But the *potential* savings from a monitoring system, in terms of both time and money, are *huge*. When something does go wrong, your monitoring system will help you pinpoint the problem quickly, perhaps avoiding a cascade of failures.

3. Which might help you predict that the data center is about to catch fire, or in retrospect, to know when the fire that raised an exception actually started.

4. The actual maintenance of system and application state is more under the aegis of tools like Cfengine [Burgess07], LCFG [Anderson08], or BCFG [Desai08].

1.3 Service Level Agreements

For some people, the primary reason to monitor is to verify that they are conforming to a service level agreement (SLA) or to help them correct problems before they are in violation of an SLA. Their monitoring systems collect hard data and alert them when they are not in compliance.

From a formal perspective, this makes sense: the needs of the business define what services are required and what quality (or level) of service is needed (the SLA—how good is "good enough"), and then systems and tools are designed and implemented to meet those needs and conform to the SLA.

One problem with this focus on monitoring for SLA compliance is that most sites don't have well-defined SLAs for all services. You might have an SLA that says that "if a file server dies, sysadmins will be notified within five minutes and the 'on-call person' will be able to start working on it within 15 minutes" (though a better SLA would be that "the file server will be up 99.99% of the time," and let the sysadmins responsible for the file server create a monitoring and alerting policy that achieves that). But who has an SLA for every service or environmental component? And, really, who wants to write them all?

Another problem is that this approach takes some of the ambition out of system and network administration: instead of aiming for perfection, it aims only for "good enough." We prefer to aim high, and use after-the-fact reporting and comparison with SLAs as a yardstick that we've achieved more than "good enough."

We like the abstract notion that SLAs and monitoring are simply a means to verify compliance, but unlike more formal, dry books, we're not going to focus on SLAs. We'll concentrate instead on enabling the satisfaction of a job well done, and the thrill of discovery—the "ah hah!" moment when you realize that there really is a reason you've been looking at all those data, charts, and graphs. Dan wouldn't have discovered (and fixed) the problems described in chapter 6 if he hadn't started his monitoring system just for the fun of it.

1.4 What Types of Data Can We Collect?

The short answer is "anything we want," but for the purposes of this booklet we are primarily interested in data that can be expressed numerically and that can be used to characterize or quantify the state of a device, service, or environmental condition.

Obvious examples are counters or gauges measuring how many mail messages have been processed or how full a disk is. But we can also keep track of less obvious data, such as how long it takes to retrieve a Web page or how quickly a mail message can be processed and dropped in a mailbox.

If you look around your environment and apply a little creativity, you can probably come up with an almost unlimited set of data you can collect and analyze. HVAC system activity, the number of telephone calls through your PBX, the load on your power bars or UPSes, the temperature and humidity in the server room, the number of pages printed on the expensive printer: these are just a few examples, and we haven't even considered computers or network equipment yet. We discuss this topic in more depth in section 2.1.

1.5 Contents of This Booklet

The remainder of this booklet is divided into seven chapters. Chapter 2 looks at the data you can collect and how you can collect it. Chapter 3 covers how you can store your data. In chapter 4 we have a look at visualizing your data using charts and graphs. We cover problem detection and notifications in chapter 5. Analysis and examples of real-world problem detection and resolution are covered in chapter 6. Chapter 7 provides pointers to software packages, hardware vendors, and other sources. And chapter 8 offers some additional advice, a summary of what we've covered, and where to go from here.

You'll notice that this booklet tends to shy away from quick and easy answers and instead presents (sometimes conflicting) alternatives. You may walk away from your initial simple question with even more questions, but we'll help you sort out the wheat from the chaff and start you down the right path to develop a monitoring solution that satisfies *your* needs and solves *your* problems.

You'll also notice that John and Dan don't always agree—and that turns out to be a good thing, because not only do you get to see divergent viewpoints, you get to see why we disagree. That will, hopefully, help you decide which of us you want to listen to, and know why your decision will be right for your problem space.

2. Data Collection

In this chapter, we look at the types of data you can collect, where the data can come from, and the different types of interfaces that can be used to actually collect the data.

2.1 What You Can Collect

In this booklet, “monitoring” means collecting and reviewing any numerical values that can change. This includes not only network statistics, but temperature, application load, connection rates, failure rates, error statistics, door status, disk status, and so on. There is a huge volume of data that a site administrator can and should collect. While syslog is often used to keep track of what various applications tell us is going on, whether that information is logging normal or anomalous transactions, monitoring is the best way to determine a qualitative baseline for your system (that is, what it looks like when everything is behaving as it should), so that you can determine when something is not right. Unlike syslog, which relies solely on the application developers to determine what is right or wrong for their subsystem, monitoring allows a site administrator to collect unrelated information (which may nevertheless include some appropriately filtered syslog data) to make data correlations. Monitoring also allows the site administrator to look at baselines and trends of data, and this can very often provide more valuable information than a simple transaction snapshot.

2.1.1 Baselines: What Is Normal?

Consider the following example: If I visit my doctor and she takes my blood pressure, she is not worried when it reads 130/85. The vast amount of statistical information that has been gathered from hundreds of millions of people suggests that this reading is normal, and were my doctor to rely on this single data point, she would be justified in giving me a clean bill of health. However, my blood pressure has always been measured at a somewhat lower value—usually around 118/78. Since we know what my baseline numbers are, we can ascertain that this reading is a little elevated. The higher-than-normal values do not in and of themselves say that there is something wrong (I might have recently eaten a fatty or salty meal or just run up a flight of stairs), but they do require us both to ask more questions: Am I under stress? What has happened to my diet? Am I exercising enough?

A baseline shows what is normal for the system (see [Burgess02]). Once you know what is normal, it is easier to determine what is abnormal. And although this sounds obvious, all too often people neglect to measure their baseline data over time and react instead to what is perceived to be an anomaly (or fail to react by failing to correctly recognize an anomaly).

Both sendmail and postfix, for example, log all incoming and outgoing email, as well as which milters have been activated for which specific messages. In this way, you can answer a user's question of "what happened to my email?" However, neither program provides a means for examining how many messages/minute have been processed, nor what is the ratio of spam to ham, nor what kinds of spam are being received and filtered. The application designers have provided a means for logging individual transactions, but intelligent monitoring provides the means for translating transactional information into statistical information—providing both baseline information and a means to note trends and anomalies.

2.1.2 Values Over Time

Essentially, monitoring allows you to travel into the past and to some degree, predict (or at least anticipate) the future. The technical term for the data produced by the monitoring we espouse is "time series data,"[1] where a sequence of data points are collected at successive (and usually uniform) time intervals. When data is available, it is possible to perform time series analysis, which can include autocorrelation or spectral analysis. Although they are beyond the scope of this booklet (and possibly beyond the scope of many of the tools we recommend), advanced signal processing techniques can be applied to all of the data you collect. Autocorrelation is a mathematical technique for finding repeating patterns, even those that may be hidden under the "noise" of a seemingly stochastically variable signal. For example, the load on your disk drives might follow a repeating pattern—recognizing the pattern might lead you to increase the amount of memory on a particular server, which would in turn reduce disk activity and increase both disk life and system responsiveness. Likewise, spectral analysis can be used to correlate Web server or nameserver load to disk activity, can suggest pages or domains most likely to benefit from local or remote caching, and can suggest where round-robin server distribution is a detriment to efficiency by causing an unduly large number of cache-misses from distributing client requests over too many server locations.

If events transpire and you don't record them, you can never go back and see what happened in the past—you can only guess at what transpired to cause the present state of affairs. But if you have kept regular records, you can go back in time as far as your records allow and see what happened. And if you know what occurred in the past, the present situation can inform the future, and you can anticipate what may come next.

2.1.3 Variations

Variations from a baseline require us to look at correlative data. Without the baseline, you can't know when something is abnormal.[2] Without correlative and historical data, you may not be able to rule out normal fluctuations (and you may not be able to discern cause and effect). Without regular monitoring, you are operating in the dark without history, statistics, trends, or the ability to observe correlations.

1. http://en.wikipedia.org/wiki/Time_series.

2. Even with statistical data, some odd readings are just "normal" for some people—I have a friend whose body temperature is always what the rest of us would consider to be a low-grade fever.

Likewise, monitoring and logging give site administrators access to data when they are not present to observe them personally. For example, assume that your site policy says, "Desktop machines should be put in sleep mode when the employee has left for the day." If you don't monitor your power consumption, how (other than walking through the office every night) can you tell if machines have been turned off? How can you tell if the cleaning crew are playing computer games at night? If you don't monitor your network load, how can you tell whether your students are running peer-to-peer file sharing? If you don't monitor temperature, how can you tell that the building manager doesn't turn off the air conditioning at night, endangering the servers in the server room?

Monitoring your system and environment on a regular basis (every few minutes) and logging that data gives the site administrator a powerful tool for not only detecting problems (and a lot of the anomaly detection can be automated), but also for visualizing normal system characteristics and seeing when things just "don't look quite right." Humans are much better than machines at some forms of pattern recognition, and taking a quick look at a "picture" of the system, network, and environmental state can often give an astute viewer an insight that a machine will miss.

Caveat: Monitoring won't fix any of your problems. It might help you detect them, prevent them, even discover ones you didn't expect, but it won't fix them—that's a different booklet.

2.2 Types of Sensor Information

There is a wealth of information available to the site administrator. Some of it is generated from the software that provides a service on your network, some is generated by hardware devices you use in your network, and some is generated by sensors whose sole purpose is to provide data. For consistency, we will simply refer to any numerical value that can be monitored as a "sensor," regardless of whether that sensor is hardware or software based.

Sensors are typically one of two primary types:

> **Gauge**: This sensor reflects the current "meter reading" of a sensor. Typical gauge sensors are temperature, door state (open/closed), humidity, load average, number of processes, etc.
>
> **Counter**: This sensor reflects the number of "units" counted by the sensor. Typical counter sensors are the number of packets or octets received or transmitted by an interface, the number of email messages rejected by the DCC[3] spam filter, or the number of ticks of an odometer. Counter sensors are time-independent (they grow "forever"), but can also typically "wrap around" after overflowing a fixed-size data storage unit (usually unsigned 16- or 32-bit integers, although 64-bit counters are sometimes used and tend not to overflow, due to their large size).

3. Distributed Checksum Clearinghouses, http://www.rhyolite.com/dcc/.

Sensors can also be one of two *derived* types:

> **Delta**: This sensor represents the number of "units" counted by the sensor in an interval (typically, since the last time it was read, although some delta-type counters are predefined as counts per minute or counts per second).
>
> **Computed**: This sensor combines two or more measurement criteria. For example, dewpoint[4] is computed using a combination of temperature, humidity, and barometric pressure. Some cup- or windmill-type anemometers (devices that measure wind speed) have a counter that tracks the number of revolutions of the spinning component. By reading the counter twice over a known period, the delta value can be measured and a count-per-minute value can be computed. This value can, in turn, be used to compute the wind velocity (given a formula for converting the spin rate into wind speed). Likewise, windchill can be computed from a combination of temperature and wind speed.

Often a data collector may provide multiple sensors that reflect related data. For example, Cisco routers provide a count of both octets (bytes) and packets. Clearly, because of the variable size of packets, the two values do not match each other, but they are certainly related.

2.2.1 SNMP

The Simple Network Monitoring Protocol (SNMP) is a popular standard for gathering statistics. Built into most modern routers and switches, as well as most operating systems, SNMP allows network administrators to examine quantitative numbers pertaining to their network. Combined with such logging and graphing tools such as MRTG, Cricket, drraw, and RRDtool (also discussed in chapter 6), SNMP can provide valuable baseline and anomaly information to a system administrator.

SNMP sensors include network load (both octets and packets) but can also include, depending on the particular device being polled, chassis temperature, memory utilization, and CPU load. General-purpose operating systems also provide access to disk utilization and free space, the number of users logged in, the number and state of processes, and more!

It is worth mentioning that SNMP daemons have had a history of security vulnerabilities. Protecting the data with a password (or "community string," in SNMP terms) is not sufficient. V1 community strings are passed in the clear, and many of the security holes are in the code that is run before or during the verification of the community string, so we recommend the following simple guidelines when using SNMP:

- ❖ Where possible, use V3 authentication (eschew V1 or V2).
- ❖ Avoid using SNMP for writing to device, and disable write access where possible.
- ❖ Block SNMP access from all but trusted hosts, both in your firewalls and in the snmpd and device configuration files.

4. The condition when the water vapor in air condenses into water—or dew—and can possibly short out electronics.

2.2.2 Environmental Sensors

Environmental sensors abound (we use the term "environmental" generally to mean anything physical in your computing or living environment). One can easily purchase (or build) sensors that monitor:

- Temperature, including cryo- and pyro-sensors for more extreme temperature ranges
- Humidity—relative and/or absolute
- Wind speed and direction (wind gust is a simple calculation)
- Barometric pressure, or pressure deltas (for clean rooms or biological applications)
- Airflow (for HVAC systems)
- Door sensors, damper sensors (for HVAC), on/off sensors[5]
- Electrical—volts, amps, watts, power-factor, kilowatt hours used, demand power, etc.
- Smoke, chemical, pH, and pollutant sensors
- Flood sensors and sonar-based water-level sensors
- Light sensors—either the light level/intensity or simply the presence/absence of light

There are a wide variety of measurement systems available for the sensors. Some manufacturers provide direct serial or Web-based access for each sensor. Others use the Dallas Semiconductor 1-Wire sensors,[6] and still others use custom analog devices (such as current-loop transformers or thermocouples) which are read by a data collector that has a networked or serial interface.

2.2.3 CPU Sensors

Many motherboards provide a means for examining CPU temperature and fan speeds. Although many systems come with redundant cooling fans, the redundancy is reduced every time a fan fails. Simply looking at CPU temperature is insufficient, as CPU temperature is often directly related to operating-system load (although it may lag somewhat behind the instantaneous system load). By examining the operating parameters of your fans and correlating this with your CPU temperature, you can know when maintenance is required, and preventive maintenance is always better than recovery from a catastrophic event.

5. See section 2.2.7 for a warning on the accuracy of binary-state sensors.

6. The term "1-Wire" means that only one wire is used for signaling. However, another wire is needed for ground, and a third is sometimes used to power some of the sensors (while others derive parasitic power from the data wire). Although sensor technology is designed to allow a multi-device bus structure, only some manufacturers' data collection hardware is designed to allow more than one device per wire. In all cases, Cat 5 cable is sufficient for wiring. The 1-Wire protocol allows up to 2,000 foot cable lengths with 100 sensors per bus. See http://www.dalsemi.com/ for more information.

2.2.4 Disk Sensors (SMART)

Most modern disk drives are SMART-capable (that is, they provide Self-Monitoring, Analysis, and Reporting Technology).[7] Although not all disks report all of the different types of SMART sensors, most drives will at least report disk temperature (usually through sensor ID #194, Temperature_Celsius). An increase in temperature may indicate cooling problems, but periodic temperature increases may simply be the result of an active disk, for example, performing a dump of a file system. Many drives will also report on the number of sectors that have been remapped (ID #5, Reallocated_Sector_Ct) and other error values (e.g., ID #7, Seek_Error_Rate and ID #201, Soft_Read_Error_Rate). An increase in these values *might* indicate an imminent failure in a drive,[8] but collecting a family of data is always more reliable than a single set of indicators from a single source.

2.2.5 UPS Sensors

Many higher-quality UPS systems provide the ability to monitor their state. This can include the quality of the line voltage and surges that have been arrested, but can also include battery capacity, estimated battery life, and various bits of information that are gathered as the UPS goes through its daily tests. By monitoring this data, you can determine when it is nearing the time to replace your UPS battery, instead of waiting for the beeping of the UPS to tell you that your battery is dead.

If you want to write your own interface to a UPS for use with MRTG, Cricket, Cacti, etc., or if you simply want to monitor your UPS without interfacing with another tool, an excellent reference can be found at the Network UPS Tools (NUT) Web site.[9] In addition to the tools, this site also keeps track of changes to the various manufacturers' communication protocols and cabling requirements.

2.2.6 Custom Sensors

What you can examine in your environment is really limited only by your imagination. Here are a few sensors we have developed by extracting information from other tools:

> **Mail load:** With the mailstats program, sendmail provides information on message queue size and volume, as well as how many messages are transmitted through each mailer. This can easily be interfaced to programs like Cricket or MRTG to be logged. However, by writing a simple daemon that continuously monitors the log file /var/log/maillog, we were able also to determine the number of deferred messages, the number of messages delivered locally versus forwarded to remote machines, the number of locally originating emails, and, most importantly, the count of each type of spam that is detected in each of the mail filters we use. From that, it is easy to see the spam-to-ham ratio on our mail

7. An introductory article about monitoring SMART-capable disks with smartmontools is [Allen04]. The smartmontools utilities provide access to SMART data from the command line—see http://smartmontools.sourceforge.net. Links to many documents on SMART may be found in the References section of the smartmontools home page at http://smartmontools.sourceforge.net/links.html#references.

8. The validity of SMART data as the sole predictor of failure is a topic of some debate: see the proceedings of the USENIX Conferences on File and Storage Technologies, and especially [Pinheiro07].

9. http://www.networkupstools.org/.

servers and quickly determine the efficacy of any new spam-blocking techniques we may install.[10]

DNS load: By writing a simple daemon that continuously monitors the BIND 9 logfiles, we are able to determine the rate of each different type of query the DNS system is handling. Because each of our redundant DNS servers is so instrumented, we can also detect when the system does an automatic failover.[11]

Web server load: With a simple script that looks at the results of the apachectl status command, we are able to look at the load on our Web server. We can show instantaneous or average query rate, bandwidth utilization, and how many processes are being used to serve user requests. Rather than guessing whether or not we have appropriate values for MinSpareServers, MaxSpareServers, and MaxClients, we can graphically examine the utilization of our Web server and configure these parameters to make best use of the available memory on the machine.

Windchill, heat index, and humidex: Although wind speed, humidity, and temperature are all physically measurable environmental units, there are derived (and subjective) measurements that can be computed to adjust the "comfort level" of a building. For example, if you observe that people tweak the thermostat after lunch, you might be able to save air-conditioning energy costs by installing a less-expensive dehumidifier to lower the dewpoint.

NTP information: By writing another small script (this one uses the system utility ntpq to acquire its data), we log the time offsets that NTP provides to each of our hosts. Not only can we see that a high volume of network traffic skews the accuracy of our time measurements, but we can quickly see the beneficial effect of adjusting the value of maxpoll in /etc/ntp.conf when talking to our local timebase.[12]

Degree days: These are an indicator of the amount of energy needed to heat (or cool) a facility. By looking at weekly or monthly degree-day figures, you can monitor the heating and cooling costs of climate-controlled buildings, and annual figures can be used for estimating future costs.

FTP information: The FTP server provides transfer and login information via syslog. By continually monitoring this information, we are able to display a metric of the volume of data uploaded and downloaded, as well as how many login attempts have failed (giving us a look at who is attempting to hack into our system).

10. A sample mail-monitoring daemon, smcount, is shown in section A.1 on p. 65; sample /var/log/mailstats output is provided in section A.2; and a method for retrieving the output for use by Cricket is presented in section A.3.

11. A sample script, querycount, is shown in section A.4 on p. 70.

12. The default value for maxpoll is 10, which makes the maximum polling interval equal to 2^{10} seconds (that is, 1024 seconds, or slightly more than 17 minutes), resulting in time values accurate to 250–750 ms. By setting maxpoll to 6 (64 seconds) or 7 (128 seconds), local time values will be much more accurate—as close as 2–10 ms. Note: only use a small maxpoll when talking to a local timeserver, though—do not overload one of the Stratum-1 or Stratum-2 servers with that many queries.

A sample Perl script for collecting NTP data into Cricket is shown in section A.7 on p. 73. Try changing maxpoll and watch the effect (although due to the convergent nature of NTP time adjustments, it may take a few days).

2.2.7 A Warning on Environmental Sensor Readings

In general, the type of data being collected should influence the sampling rate, or how often the sensor is read. For more slowly changing values such as temperature and humidity, reading the sensor every minute or so is sufficient.[13] However, for quickly changing or transient values (such as door open or closed status), it is possible to miss a status transition if the frequency of measurement is less than 2x the frequency of change (i.e., the Nyquist frequency).[14] For example, if the status of a door is measured once a minute and it takes a person less than 10 seconds to open a door, enter a room, and close the door, it is very possible that the state transition will go unnoticed. In the language of computer security, this is a "race condition," which needs to be avoided. You can sometimes sidestep these kinds of problems if your sensor provides a counter of the number of state changes, or an indication of the time of the last state change, as we discuss in the following section.

Advice: If the *changes* in an environmental or binary-state sensor are important to you (as opposed to simply fetching the current state when reading a value), be very careful when choosing your hardware. To promptly detect changes in sensor state, you need an asynchronous (or asynchronous limit alerting) sensor, described below. These are descriptive (not industry-standard) terms, so make sure you understand the differences when you ask your hardware vendor questions, so that you get the answers you need.

2.2.8 A Warning on Monitoring System Load

Be careful when monitoring system load average. Some of the tools we describe are very lightweight, but others can easily affect the system load you are monitoring. If your tools exert a sufficiently large load on the system, you may be hiding the true load imposed by other applications.

Load average is just one of many characteristics that you can monitor on a system—by taking care not to depend on a single statistic, you can continue to gather meaningful information about your system status.

2.3 Underlying Sensor Hardware

When choosing a data collector, and especially when measuring binary states with dry-contact switch sensors,[15] you need to be aware of what kind of sensor is used by the underlying hardware.

13. Most devices we have encountered physically poll the sensor only when you ask for data. However, some devices poll the sensors independent of user-based queries, and return the most recent (or the recent average) reading when a request is made. The OWFS family of interfaces can also cache recent readings and will simply supply a recent value if two queries come in quick succession (although a means is also provided to bypass the cache and provide raw readings at every query).

14. http://en.wikipedia.org/wiki/Nyquist_frequency.

15. A "dry contact sensor" is simply a sensor that can read the open and closed state of an external switch.

There are typically five classes of binary sensors:

> **Simple**: A simple sensor returns the binary state when it is queried, and provides no history of past events.
>
> **Latching**: A latching sensor returns either the current value, or the difference between the previously read value and the current value if a change has occurred. For example, a latching door sensor will return "closed" if the door is and was previously closed; it will return "open" if the door is open and was previously closed; and it will also return "open" if the door is currently and was previously closed, but if at any time between the previous reading and this one, the door was opened.
>
> **State Preserving**: A state preserving sensor is a combination of a simple sensor and a latch. In other words, when the sensor is queried it returns the current binary status and latches any change (so, for example, you can see that the door is presently closed but was opened at some time between this reading and the previous one).
>
> **Asynchronous**: Asynchronous sensors return the current state when queried, but also provide an asynchronous event (such as an SNMP trap, email, or SMS) when the binary state changes. Asynchronous sensors are best for alarm systems, where it is important to know that an event (such as a door opening or a smoke alarm being activated) has *just happened*.
>
> **Logging**: Logging sensors keep track of when binary state transitions have occurred, so that when the sensor is queried, the current and the timestamped historical state can be ascertained. Logging sensors also include state indicators (i.e., open or closed) backed up by a counter (number of transitions).

Unfortunately, the adage of "you get what you pay for" does not necessarily apply to sensor technology. Some of the more expensive (and well marketed) data collection devices we know of have only simple binary sensors. If your environmental monitoring application needs to know that (or when) a state transition has occurred, even if it is between measurement intervals, then take care when selecting your hardware.

The "slower" sensors (i.e., the kind whose values change more slowly, such as temperature, power consumption, and humidity) also come in four basic flavors:

> **Simple**: A simple sensor returns the device state when it is queried and provides no history of past events.
>
> **Min/Max**: A min/max sensor returns the device state when it is queried, but can also provide the highest and lowest readings for the sensor. This is most useful when the device polls the sensor independent of the software that queries the device (so you can safely poll the device every few minutes, but still be apprised of a 30-second spike between user readings).
>
> **Asynchronous limit alerting**: Asynchronous limit sensors return the current state when queried, but also provide an asynchronous event (such as an SNMP trap, email, or SMS) when the sensor value exceeds some predetermined limit. Asynchronous sensors are best for critical infrastructure or alarm systems, where

it is important to know that an event (such as temperature in a server exceeding a safe limit or water in a sump rising above a threshold) has *just happened*.

Logging: Logging sensors keep track of when binary state transitions have occurred, so that when the sensor is queried the current and the timestamped historical state can ascertained.

Note that even with the min/max or asynchronous sensors, it is still possible to miss critical events. If the underlying sensor provides a voltage or current output based on sensor value (such as thermistor or thermocouple), the hardware can be constructed to instantaneously detect high or low sensor values. However, if the sensor is digital and must be read to determine the sensor reading (for example, any of the 1-Wire sensors), then even the low-level hardware data collector must poll the sensor and can miss short-lived events. If your monitoring application requires a fine temporal resolution, take care when selecting your hardware. Realistically, very few applications of environmental monitoring require that degree of resolution. With the exception of binary sensor status (above), you are probably safe with simple sensors that you query every minute or so.[16]

2.4 Types of Interfaces

There are a wide variety of data gathering and configuration interfaces to sensor devices; some devices provide multiple interfaces, and some of these separate the functions that can be performed on each interface. Typically, however, interfaces fall into one of two categories: Ethernet TCP/IP protocol-based interfaces, and some form of serial communication protocol (RS-232, USB, MODBUS, etc.). We will attempt to enumerate the commoner ones we have seen.

2.4.1 SNMP

SNMP (the Simple Network Monitoring Protocol) is the standard mechanism for querying the counters and gauges found in most modern network appliances. Almost every router, switch, and wireless access point implements a MIB[17] that defines what values are accessible through SNMP, and most of them use standardized MIBs for common values (e.g., `IF-MIB::ifInOctets` or `IF-MIB::ifInUcastPkts`[18]). Just about every modern operating system also provides an SNMP daemon for monitoring the state of the system.

SNMP data is accessed through the SNMP protocol, and almost all of the monitoring tools have SNMP support built into them. However, there is a large collection of command-line tools (snmpwalk, snmpget, etc.) and development libraries available for anyone who wants to write their own SNMP monitoring tools or simply look at the current status of an SNMP variable.

16. Often we query our sensors every minute but only log the 10-minute average values.

17. A MIB, Management Information Base, defines the names of the attributes that can be monitored and also defines their "addresses" (called Object Identifiers, or OIDs). By "walking" the MIB using SNMP, you can read every data value, but it is more common for applications to selectively query specific OIDs to extract the desired information. The data that is most often queried through SNMP is network parameters, such as number of packets, octets, or errors that have been encountered on a collection of interfaces, but SNMP can also be used to collect environmental information (such as temperature and humidity) if the device has the appropriate sensors and the appropriate OIDs defined.

18. These notations mean "in the Interface MIB, interface Inbound Octets (i.e., bytes) and interface Inbound Unicast Packets" (as opposed to outbound octets or packets).

2.4.2 HTTP

By far the most common interface to environmental sensor data collection devices is HTTP. Some devices allow you to specify parameters (either GET or POST, depending on the sophistication of the device), while others provide different page addresses to gather data, and still others provide information on all sensors in a single page. Very often the printed documentation for a device will specify a page that provides a Java interface to the device, but by examining the HTTP traffic using tcpdump or wireshark it is possible to determine what underlying HTTP request is being made (and that can then be queried by automated data collection tools).

Most devices provide a pretty HTML interface where the sensor values are arranged in tables for easy viewing. However, certain manufacturers also provide a much simpler interface when they anticipate automated data collection tools (or to provide a simpler debugging interface for their developers). While some manufacturers will freely tell you what these simpler interfaces are (either in their printed documentation or with a call to tech support), they tend to be very protective of their "proprietary interface." Judicious traffic sniffing (while using their "proprietary monitoring tool") will show you the necessary requests and responses to collect the data you need.

The formats provided by data collection devices can be as simple as CSV or some form of keyword=value. Other devices provide a URL that responds with an XML-formatted version of the data (see section 2.4.8).

```
{
 name:"RMA-71184",
 date:"02/14/37 23:48:28",
 scale:0,
 sensor:[
  {label:"Basement",
   tempf:"73.40",tempc:"23.00",highf:"77.18",highc:"25.10",
   lowf:"54.68",lowc:"12.60",alarm:0,type:38,enabled:1,
   humid:"46.58",highh:"54.06",lowh:"21.42"},
  {label:"Attic",
   tempf:"76.55",tempc:"24.75",highf:"90.95",highc:"32.75",
   lowf:"62.15",lowc:"16.75",alarm:0,type:16,enabled:1},
  {label:"Sensor 4-1",
   tempf:"32.00",tempc:"00.00",highf:"32.00",highc:"00.00",
   lowf:"32.00",lowc:"00.00",alarm:0,type:0,enabled:0}  ],
 switch_sen:[
    {label:"Door",alarm:1,status:0,enabled:1},
    {label:"Airflow",alarm:1,status:0,enabled:1},
    {label:"Flooding",alarm:1,status:0,enabled:1},
    {label:"Sensor 1-4",alarm:1,status:0,enabled:1},
 ]
}
```

Figure 2.1: Sample RoomAlert JSON Notation

However, some can be more complicated and may use nonstandard formats. For example, the RoomAlert[19] series of data collectors uses a hierarchical (but easily parsed) JSON notation,[20] as shown in figure 2.1 (with spaces added for readability).

```
<table name="Temperature" id="Temperature">
<tr>
<td colspan=1>Address</td>
  <td colspan=1>Temperature</td>
  <td colspan=1>Resolution</td>
  <td colspan=1>Status</td>
</tr>
<tr>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Address_0"
    ID="Address_0" TYPE="text" VALUE="71000800C28E5710"></td>
  <td colspan=1><INPUT CLASS="HA7Value"
    NAME="Temperature_0" ID="Temperature_0" TYPE="text"
    VALUE="20.4375"></td>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Resolution_0"
    ID="Resolution_0" TYPE="text" VALUE="9+"></td>
  <td colspan=1><INPUT CLASS="HA7Value"
    NAME="Device_Exception_0" ID="Device_Exception_0"
    TYPE="text" VALUE="OK"></td>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Device_Exception_
    Code_0"
    ID="Device_Exception_Code_0" TYPE="hidden" VALUE="0"></td>
</tr>
<tr>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Address_1"
    ID="Address_1" TYPE="text" VALUE="97000800E8113E10"></td>
  <td colspan=1><INPUT CLASS="HA7Value"
    NAME="Temperature_1" ID="Temperature_1" TYPE="text"
    VALUE="21.375"></td>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Resolution_1"
    ID="Resolution_1" TYPE="text" VALUE="9+"></td>
  <td colspan=1><INPUT CLASS="HA7Value"
    NAME="Device_Exception_1" ID="Device_Exception_1"
    TYPE="text" VALUE="OK"></td>
  <td colspan=1><INPUT CLASS="HA7Value" NAME="Device_Exception_
    Code_1"
    ID="Device_Exception_Code_1" TYPE="hidden" VALUE="0"></td>
</tr>
</table>
```

Figure 2.2: Sample HA7Net HTML Output

19. http://www.roomalert.com/.

20. JavaScript Object Notation, http://json.org/.

Some HTML-interface devices intermingle the "pretty" output that a human can read with the data needed for automatic parsing (such as a DOM or SAX parser).[21] The HA7Net interface from Embedded Data Systems[22] produces HTML that looks like that shown in figure 2.2 (whitespace added for readability).

Since each reading has an ID and a VALUE, this output can easily be parsed by reasonably simple regular expressions in addition to the more sophisticated DOM and SAX parsers.

However, when all else fails and a device generates irregular HTML, it will be necessary to parse whatever unconstrained HTML is output by the device. In this case, looking for patterns and special delimiters is your best bet to discern the information you need. It is sometimes also useful to use some form of HTML formatter, such as the lynx text browser, and format the HTML before trying to parse it.

2.4.3 OWFS—1-Wire File System

The 1-Wire File System[23] is a set of kernel modules that provides a seamless integration of the Dallas Semiconductor 1-Wire devices into most operating systems (and some embedded appliances) using a large variety of bus masters,[24] including USB, serial, and Ethernet adapters.

We call out OWFS specially here because it provides a simple means for accessing environmental sensor readings using three different interfaces:

> **owfs**: File system access to sensors (e.g., simply "cat"ing the file named /owfs/10.001122334/temperature would give the temperature reading from the OWFS sensor with the serial number 10.001122334).
>
> **owhttpd**: Web access to sensor (owhttpd creates a mini Web server that accesses the same set of sensors that are available locally through owfs).
>
> **owserver**: Remote network access, using a well-defined network protocol and a set of command-line commands, provides remote access to the same set of sensors that are available locally through owfs.

OWFS also provides direct language support for Perl, Python, PHP, Tcl, Java, and C, and as such can provide a very easy means for multiple sampling points with a centralized data-collection system.

2.4.4 MODBUS

"MODBUS is a serial communications protocol published by Modicon in 1979 for use with its programmable logic controllers (PLCs). It has become a de facto standard com-

21. DOM and SAX are two methods for representing and manipulating objects in tagged text, such as HTML or XML.

22. http://www.embeddeddatasystems.com/.

23. http://owfs.org/.

24. http://owfs.org/index.php?page=bus-masters/.

munications protocol in industry, and is now the most commonly available means of connecting industrial electronic devices."[25] The protocol is freely available and royalty-free.

There are two different MODBUS communications protocols:

> **ASCII**: In ASCII mode, the communication characters are all printable ASCII, and simple checksums are used for data integrity.
>
> **RTU**: In RTU (Remote Terminal Unit) mode, the communication characters can be any ASCII character, and CRCs are used for data integrity.

According to the MODBUS spec, every device must implement RTU mode and may implement ASCII mode as an option. For debugging purposes ASCII mode is clearly easier, but may not be available.

In addition to purely serial communications (either over RS-232 or RS-485), the MODBUS spec also allows for MODBUS over TCP/IP. This is more than simply a wrapper around the serial protocol, and introduces better error handling, but also requires a different protocol stack to use than serial.

There are MODBUS protocol stacks written in C, C#, Perl, Python, and many other languages. The actual protocol is not that difficult, and if you want to write your own custom monitoring systems, it is relatively easy to implement a MODBUS master to query any number of devices.[26]

2.4.5 ASCII Serial

Some devices simply communicate through a serial-line interface, usually RS-232 but occasionally RS-485. RS-232 devices can only be connected to a single computer, and thus that computer must act as a gateway for any other computer that wishes to get data from a serial device. In theory, multiple computers and/or devices can be connected to an RS-485, but in practice the same restrictions apply to RS-485 as to the RS-232.

It would be nice if all serial devices adhered to official communications and connector standards,[27] but some manufacturers require custom cables. Notable in this cacophony of non-standard connectors are UPS systems from APC[28] and routers and switches from Cisco (the latter, fortunately, provides a reliable Ethernet communication protocol in addition to the serial line interface).

25. http://en.wikipedia.org/wiki/Modbus.

26. In fact, the only tricky bit about the MODBUS spec is the terminology: "coils" are single-bit on/off values, and "registers" are 16-bit values (which can, in some hardware implementations, be combined to provide 32-bit or larger values). Coils and registers are sometimes read-only, but can also be read-write, and one oddity of the protocol is that different addresses are used to read and write. Another oddity is that registers are numbered 1..N, but the protocol addresses them as 0..N-1.

27. For examples, see http://en.wikipedia.org/wiki/RS-232 or http://www.loop-back.com/rs232_std.html.

28. Standard DB-9 wiring places TXD on pin 3 and RXD on pin 2. However, APC's smart serial ports put TXD on pin 1 and RXD on pin 2. If you are not handy with a soldering iron, it is worth buying a pre-wired APC Smart Signaling cable or equivalent.

To add to the complication of serial device protocols, some devices use the absence of DTR and RTS to reset the device to a known initial state, while others preserve the pre-existing state of the device between connections.

Most devices use some form of ad hoc data representation when communicating sensor values. While it would be nice to imagine that all devices use some form of CSV or `keyword=value` format, the truth is that there are probably almost as many data formats as there are devices. Line delimiters may be newlines, carriage returns, or both possible combinations of the two (i.e., `<CR><LF>` or `<LF><CR>`), and field delimiters may be spaces, commas, semicolons, colons, etc.

Serial devices fall into two broad categories:

> **Polled**: Polled devices must be asked for data. A query (which may be anything from a single character to a complex query string) is sent to the device, which then replies with the value(s) you have requested. The biggest problem with polled serial devices is not in handling data responses, but in handling erroneous responses. Very often, the state diagram for serial devices adequately describes the normal response, but documents the error states poorly or not at all—and recovering from an error state is something that often requires a human's intuition. Programmatic recovery from an error can sometimes be best accomplished by resetting the device (shutting and reopening the serial line connection may be sufficient, but often the best approach is a hard reset of the device).
>
> **Streaming**: Streaming devices send their data on a periodic basis. Some devices poll their sensors and send the data at regular intervals (once a second, once a minute, etc.), while others send data as fast as it is read from the sensors. In this latter case, whatever software you use to read the data must take care to extract it from the serial line fast enough that the kernel buffers do not overflow. While loss of data is not necessarily a problem, the varying behavior of various operating systems on buffer-full conditions is. Some will signal to the device by dropping CTS until the buffer is emptied (and the response to CTS and DTR in the device itself is equally unpredictable), while others will simply drop characters, in which case re-synchronization with the input data stream may be a problem, as different devices use different end-of-record separators.

In general, we have found streaming serial devices to be easier to deal with, but even well-behaved devices have their problems.

2.4.6 Telnet

Some devices provide a Telnet port instead of a serial port, so that standard Ethernet wiring can be used instead of custom (and more limited) serial lines. The same broad categories of interfaces (polled vs. streaming) apply, as do the same oddities regarding state preservation between Telnet sessions.

2.4.7 USB

Many devices provide a USB interface instead of a simple serial interface, and the biggest problem is reading the data from the device. The only thing "universal" about the

Universal Serial Bus is the electrical signaling and low-level packet formatting. The higher-level data exchange is entirely device dependent, and unless you have a driver for the specific piece of hardware, you may be unable to communicate with your device. We have encountered USB-based temperature and humidity sensors, electrical monitors, and even USB oscilloscopes; however, all require custom drivers, the majority of which only run on Windows.

Most of the USB monitoring devices also provide some kind of logging and graphing software, but you are typically constrained to the kind of information recording that the designers envisioned. For most of the devices, you can run a logging tool, but then you must manually save the recorded data to a CSV or Excel file, and extract it later using other tools. Some of the installation packages for these devices also provide a command-line interface to read data from the device, in which case you can automate your monitoring (beyond what the device manufacturer envisioned), as described in section 2.4.9.

However, if you want to use USB for environmental monitoring,[29] OWFS (section 2.4.3) provides a relatively seamless interface between the 1-Wire File System and the Dallas Semiconductor DS9490 USB-to-1-Wire adapter.

2.4.8 XML, XML-RPC

Some devices communicate data requests and responses using XML-RPC over HTTP. Although not a communication standard itself, XML is used as the encapsulation method for XML-RPC and is also used for some devices to communicate data values. XML is fairly straightforward, and some devices provide a means of extracting data from the device in XML format. For example, if the Poseidon series of data collectors[30] is queried via HTTP for the page named values.xml, the output will be an XML encoding of everything that can be found on the Flash-based Web page (that is, both a machine-readable and a human-readable interface are provided).

An example[31] of a typical XML-RPC request would be:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

An example of a typical XML-RPC response to this query would be:

```
<?xml version="1.0"?>
<methodResponse>
```

29. Many hardware appliances only have a USB port and no longer have a serial port.

30. http://www.hw-group.com/.

31. See http://en.wikipedia.org/wiki/XML-RPC for more details.

```
  <params>
    <param>
      <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

To use XML-RPC to query data collectors, an XML query must be constructed (using the device-defined element names) and transmitted to the device (typically using HTTP as the transport protocol). The device will then respond (using either positional or keyword/value parameters) with an XML message that contains the desired data values. Although the process is rather cumbersome to describe and implement, XML-RPC protocol stacks exist in C++, Perl, PHP, Python, Java, and many other languages, and the API is considerably simpler than the protocol.

2.4.9 Command Line

Finally, many sensors and/or data streams are built into the computer hardware, operating systems, and/or application programs. They therefore have command-line interfaces which provide information, typically in a human-readable format (which is often readily parsed using simple regular expressions). By using existing programs, you are spared writing all but the simplest wrapper code and are given access to a wide collection of valuable statistics. Some examples include:

- ntpq, the standard NTP query program, reports on the offset and jitter of network time sources.
- mailq reports on the contents of the mail queue, allowing you to track the current size of the queue.
- mailstats reports on the total number of messages and bytes transferred by each mailer agent.
- sysctl reports on a large number of kernel variables for status and statistics gathering. For example, look at the output of `sysctl -a | egrep thermal` or `sysctl -a | egrep vm.stats` for just a taste of the wealth of data that sysctl can provide.
- smartctl is a program which can monitor SMART disks. See section 2.2.4 for more details, or try the command `smartctl --attributes /dev/ad0`, replacing /dev/ad0 with your disk name.
- dmidecode[32] provides access to the computer's DMI (Desktop Management Interface) table, which supplies information on the hardware and its state.
- The FreeIPMI[33] tools provide command-line access to IPMI (Intelligent Platform Management Interface) information, for monitoring system health and managing the system, through the baseboard management controller (BMC).
- cyradm is the Cyrus mailer administrative program. With the `info` command and appropriate wildcards, you can monitor the size of every user's mailbox to detect disk hogs or influxes of spam.

32. http://www.nongnu.org/dmidecode/.

33. http://www.gnu.org/software/freeipmi/.

- apachectl, the Apache Web server status command, can track the volume of network traffic the Web server is taking (both in terms of bytes and requests) and can also report on the number of active requests.
- ping can determine network latency (which may vary greatly, even when your network throughput is guaranteed).
- nfsstat, vmstat, iostat, fstat, etc., can all be mined for useful information.

3. Data Storage and Logging

In chapter 2 we looked at what kinds of data you might want to collect, and ways in which you could collect the numbers you're interested in. In this chapter we're going to discuss how you can store the data you've collected so that it's available for analysis.

Before deciding how we're going to store our data, it will help if we have an idea of how we might want to use the data, i.e., what sorts of analysis we might want to perform on the data. If we're just going to look at the most recent numbers, we may store (and retain) the data one way; if we're planning to draw graphs or other visualizations of our data, we will want to store it in a way that makes it easy to do that.

We will also want to look at how much data we store, for how long, and where we store it. Do we store everything we collect, or only those data points that we think might be useful? Do we keep the data forever, do we aggregate individual data points into averages or other summary numbers, do we throw away old data? Do we store the data where we collect it, or do we gather it up centrally for easier analysis and aggregation?

3.1 What to Keep, What to Toss

When you're collecting data, you may end up collecting more information than you really want to keep. For example, you may be collecting BIND statistics from your authoritative (non-resolving) DNS servers and may not be interested in retaining all the zeros that represent the recursive lookups you're not doing. Or you may be collecting syslog records from your mail servers to determine traffic levels and aren't interested in the details of who is sending what to whom.

If we may be permitted a broad generalization, we'll divide people into two points of view:

Minimalists: Keep only the bare minimum of what they need, and avoid overwhelming themselves with too much data.

Packrats: Keep everything that they can possibly collect, just in case they might be able to use it some day.

We'll offer what we claim is practical advice: you're generally better off leaning toward the minimalist rather than the packrat approach.[1] Sure, there is a possibility that some

1. Dan says: I have Web server logs going back to 1995 for all of my Web sites (and that has been valuable source material for two research projects). If you say "the weather feels cold for this time of year," I can authoritatively tell you what it was like last year (and the year before, and the year before . . .). I reluctantly throw out old paper, but disk space is cheap (and I only have a half-dozen gigabytes of well-organized, compressed data). Fellow packrats have helped me bust bogus software patents when they mailed me copies of 15-year-old mailing list archives. Log data is irreproducible, and well worth keeping. I freely confess to being a packrat—and it has served me in good stead!

day you might want to look up some information that you didn't keep, but if you have a non-trivial environment, keeping everything can quickly become overwhelming. Remember the old saying about not being able to see the forest for the trees? Keeping too much is perhaps more like not being able to see the tree you need because the forest is completely overgrown.

Let's have a look at some things you should consider when deciding what data you should keep:

> **Characterization vs. Detail**: Consider whether you need to retain all possible detail, or if you really just need enough information to characterize your data. On your Web server, for example, do you need to track the exact browser versions that visited your site, or is a broad characterization (Internet Explorer, Firefox, Safari, etc.) sufficient? Can you aggregate the data in a way that gives you the information you need and throw away the details? For example, can you keep detailed information for recent time periods, and keep only averages of older data? If you aggregate or summarize, do you still need to keep the details so that you can "drill down" into the detailed records?
>
> **Policy and Legal**: Depending on your industry (and jurisdiction), there may be strict legal or policy requirements on what data you can keep and how you can keep it. For example, if your organization is involved in health care or finance, there may be requirements on whether you can (or must) store identifiable information (e.g., if you're storing and analyzing mail logs). We obviously can't offer a definitive answer here, but we wonder if, when considering monitoring data, you're better off in the "less is more" camp in these situations.[2]
>
> **Different Environments**: If you have different "levels" in your environment (development, test, and production, for example), you may want to collect and retain different amounts of data in your different environments; set different rules for external and internal services; be more careful about the performance of the "public face" of your organization. You may want to adjust your retention based on recent experience: if one of your environments has been having more problems or has higher visibility in the organization, you might want to keep more, or more detailed, data.
>
> **Got Space?** Collecting and storing data requires system capacity (CPU, memory), network bandwidth,[3] and disk space. These days we often say "disk is cheap," but the more data you collect, the longer you retain it, and the more systems and devices you have, the more space you will need (obviously). And with more data, you'll need more processing capacity in order to analyze and report on your data (and, some might argue, the more data you have, the more likely you are to just spend a lot of time "playing" with the data). This of course is balanced out by the fact that most monitoring data is relatively compact, especially when compared to some of today's larger data sets. And, as we shall

2. WANAL—We Are Not A Lawyer, your mileage may vary, void where prohibited, please check with an authoritative source before proceeding.

3. Except in rare cases when you might be storing system monitoring data locally on each system. For an interesting project which addresses both local and remote storage and analysis of data (complete with anonymization of sensitive local information), look at the EDDY project from CMU: http://www.cmu.edu/eddy/.

see in the next section, there are many different ways to store monitoring data, which are more or less space-efficient, depending on the storage method.

Got Bandwidth? Another element to consider is the network bandwidth used for collecting monitoring data. Again, in modern terms on modern networks, the bandwidth used to collect monitoring data is usually going to be inconsequential, but you may have remote sites or other special cases where network bandwidth capacity is a limiting factor. There is also the question of disk I/O bandwidth, which can sometimes become a limiting factor. As the amount of data collected and stored goes up, storing lots of little bits of data, in lots of small files, can add up to lots of blocks going to and from the disk.[4]

There's obviously no one-size-fits-all answer, but hopefully these discussion points can help you decide what makes sense in your environment.

3.2 A Place for Everything

There are many different ways to store monitoring data, but some methods are specific to monitoring (or other time-series) data, and some work especially well (or poorly) with large amounts of small bits of data.

3.2.1 Flat Text Files

There's a long history of success and ease of use for data stored in simple, flat, ASCII text files.[5] Tools such as awk, Perl, and the shell provide easy and effective ways to manipulate and analyze data stored in text files. And writing collected data to a text file is often trivially easy. However, this simple ease of use can mask some potential problems:

- Data must often be searched or processed sequentially.
- Text files may not be the most space-efficient way to store monitoring data.

But these potential problems can usually be mitigated after the fact, by converting the text files into another format.

Text files can be organized (or disorganized) in many different ways, depending on the data being stored and how you expect to access the data. The most common methods typically use one text line for each "record"—a data point or a set of data points, collected at the same time. This is handy in a number of ways: the line count equals the record count, you can easily access each record in turn, and you can easily split a large file into smaller files, if desired, without worrying about splitting a single record across multiple files.

The data in a record (a row or line) can be organized in several ways:

- White-space-separated values in a known order
- Fixed-size columns of data, again in a known order

4. It's likely worth mentioning that current versions of RRDtool (section 3.2.2) have special features to reduce the I/O required to update the RRD files.

5. Dan also confesses to being a big fan of flat-text files—he can manually edit erroneous data, correct errors, and write quick-and-dirty scripts to process data. He also admits to being something of a Luddite when it comes to data storage.

- CSV (comma-separated values) format, accepted by spreadsheet programs
- A "native" format (e.g., syslog output)
- Some form of tagged format (e.g., `keyword=value`)

You can of course create your own file format, although it will be helpful if your chosen format is "well-defined," to make it easier to understand in the future.

There are likely arbitrarily many multi-line record formats that you could choose (or invent). The most popular these days may well be text tagged with XML (or XML-like) tags. XML can be generated as a single line per record, but it's more commonly rendered in an easy-to-read indented format. Two obvious downsides to XML (which may or may not apply in your particular case):

- There is typically a non-trivial parsing overhead for XML, which may become significant when dealing with non-trivial numbers of records.
- There is almost always a non-trivial space overhead for the tagging itself, and, in the case of monitoring data, the tagging may well require far more file space than the data itself.

Our opinion: XML is not a good choice for storing monitoring data. People will use it anyway because it is the "in" thing to do, but the signal-to-noise ratio is too low and the processing overhead is too high for it to be the format of choice.

It's likely worth pointing out that even the tersest of text files will take up more space than the equivalent data stored in a binary encoded format (e.g., four- or eight-byte floating point numbers). A binary file format will save space but will typically be less convenient to manipulate, and may not be portable across different operating systems or hardware platforms. Compressed text files save space, but then you pay the price of decompressing them when you need to read the data. However, modern decompression software is fast, and for data that is always read sequentially and not accessed "often," compression is a reasonable compromise between text and binary formats.

As mentioned above, a major benefit of text files is that there are so many convenient tools for manipulating them. If you're using text files, you will likely want to use multiple files, perhaps one file per day (or hour), or separate files for different devices or types of data. In addition to the tools already mentioned, it is worth also mentioning:

logrotate: Logrotate[6] provides an easy mechanism to periodically rename ("rotate") log (or other) files and, optionally, to compress the files to save space.

cronolog: Cronolog[7] provides a method to easily write your log (or other) data into files named for the current date and time, so you don't need to rename or rotate your log files daily or hourly.

6. https://fedorahosted.org/releases/l/o/logrotate/.

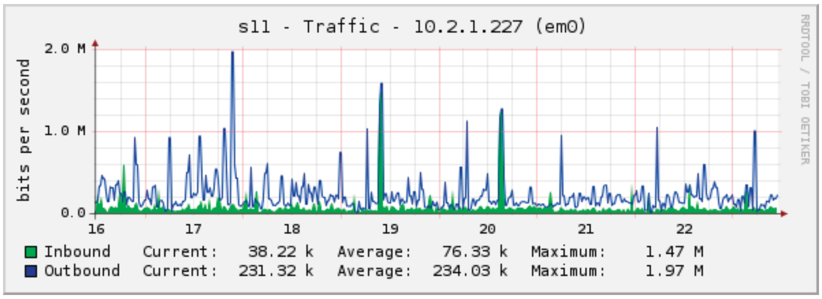
7. http://cronolog.org/. John recommends that you save your Web server logs by piping them through cronolog.

3.2.2 The Miracle of RRDtool

RRDtool[8]—Round Robin Database tool—is a very popular system for logging and graphing time series data; it was originally developed for storing network monitoring data. Underlying RRDtool is the RRD file format, which provides a space- and time-efficient way to store numeric data over time and offers an easy way to aggregate or summarize older data.

In typical use, an RRD file is configured to store all recent data points, but to store only consolidated data (averages, maximums, minimums, etc.) further back in time—perhaps hourly averages over the past week, daily averages over the past month, and weekly averages over the past year. RRDtool, either from the command line or through various language APIs, is used to update the RRD files with new data or to access the data by either generating graphs or extracting the numeric data for other uses.

Figure 3.1 shows a typical graph that was created by RRDtool and we'll learn more about graphing with RRDtool in section 4.3.4. (If you have a printed copy of this booklet, please note that RRDtool-generated graphs are typically created in color.)



```
🟩 Inbound   Current:  38.22 k  Average:  76.33 k  Maximum:  1.47 M
🟦 Outbound  Current: 231.32 k  Average: 234.03 k  Maximum:  1.97 M
```

Figure 3.1: A Typical Graph Generated with RRDtool

There are two key ideas about typical use cases that form the basis for RRDtool:

- ❖ You often want details of recent activity, but only a "characterization" of older data.
- ❖ Monitoring data is collected far more often than it is analyzed.

One aspect of RRDtool is that you have to plan ahead, and it helps if you are (at least somewhat) consistent in what data you collect and how often you collect it. Before you collect any data to store in an RRD file, you must explicitly create the file. When an RRD file is created, you must specify what data you plan to store there, how often you expect to update it, how long to keep the data, and how the older data should be aggregated (through a "consolidation function": average, maximum, minimum, etc.), and the RRD file is created with a fixed size. You can of course create RRD files as needed (such as when a new device is discovered on your network), but the point is that you can't just throw all your data into a pile and expect to deal with it later.[9]

8. By Tobi Oetiker, winner of the SAGE 2006 Outstanding Achievement Award for his work on MRTG and RRDtool: http://oss.oetiker.ch/rrdtool/.

9. Well, actually, you can, if your future plan is to tidy up the data and properly add it to RRD files. But you'll be better off if you do things "right" the first time.

3.2.3 Traditional Database

The "traditional" method for storing large amounts of data is to use a general-purpose relational database. A database system provides easy (or at least well-defined, well-known, and consistent) ways to access the data, select what you're interested in, and manipulate large amounts of data with relatively simple commands. If you already have a database infrastructure in place, perhaps you can rely on someone else to maintain the storage space and processing power you will need. The biggest problem is that traditional databases are not optimized for constantly appending data. Locking, indexing, and other housekeeping functions done by SQL servers are superfluous to monitoring systems. This extra work makes it hard for traditional databases to scale.

There are a number of monitoring tools that store the data in a relational database,[10] but this may not be the best approach to data storage in many cases.

- A relational database provides significant functionality, but there is an overhead cost to providing these features. This may lead to greater resource (CPU, memory, disk) usage being required to store and manage monitoring data.
- If you are collecting large amounts of monitoring data, the time required to complete a database insert may add a significant amount of "wall-clock" time to your periodic data collection. If you're collecting often (every few minutes), these small bits of overhead may add up to a significant problem.

In many cases, you are likely better off using some other method to store your monitoring data, unless you're already database inclined and enabled and it's already your standard method for storing and accessing data.

3.2.4 Non-Traditional Database

For some of the benefits of a traditional, full-blown database system with less overhead, you can also look to non-traditional databases. By "non-traditional" we mean a database that is not general-purpose but tailored to a specific use.We'll mention two here, but there are others. One could also say that the file system is a special form of database as well, so just about anything could apply here.[11]

Berkeley DB[12] is a non-relational database, which stores key/value pairs in a database file with very low overhead, easy access, and full database functionality. It is widely used where quick and easy data access is required, especially in embedded systems and in places such as sendmail and passwd file hash tables. With some creative ways of generating names for your files and database keys, you could store your monitoring data in Berkeley DB files.

SQLite[13] is a library that implements a SQL database in a file. Somewhat similar in concept to Berkeley DB, SQLite provides additional functionality, with multiple columns,

10. For example, Real Traffic Grabber (RTG) [Beverly02].

11. There's an easy argument to make that RRDtool also falls into the non-traditional database category.

12. http://www.oracle.com/database/berkeley-db/index.html.

13. http://www.sqlite.org/.

tables, joins, and all the things you would expect in a relational database, but with less overhead and almost no management or configuration.

But we can't help thinking that if you're looking for an easy, efficient, and flexible method for storing your monitoring data, RRDtool is very likely a better choice than another one of the non-traditional database alternatives.[14] And if you don't like the idea of RRDtool consolidating your older data, simply create your RRD files so that they don't do any consolidation.

3.3 Here a Point, There a Point

When you collect your data, you may be doing it on the local system (i.e., where the data originates) or from some sort of central monitoring server (a "management station," in SNMP parlance). Regardless of where you do the collection, you're going to have to decide where to store it.

If you store the data on the local machine (assuming the data is from a computer rather than from something like a network switch or router), you'll know where everything is, you'll have it handy when you're looking at a particular system, and you won't need to worry about getting resources allocated for a central server to store everything. But:

- It will likely be harder to access from a central console.
- It will be harder to aggregate and provide summary information across a collection of devices.
- You'll essentially be mixing your "overhead" data (the monitoring data) in with your "production" data.
- You'll have to manage the data (section 3.3.1) in multiple locations, which will likely add some complexity.
- Anytime you're looking at how a system has been performing, you'll be adding additional load onto the very machine you're already concerned about.

Both John and Dan lean toward the "centralize your monitoring data" school of thought, for ease of access and management.

Depending on your network, organizational structure, and how many machines and how much data you have, a hybrid or tree structure may make the most sense. By this we mean storing your monitoring data based on geographical or organizational boundaries (e.g., one monitoring server in Asia, one in North America, one in Europe, or one in the head office and one in each data center). You can of course use various hybrid structures, such as distributing your data collectors but centralizing the data (recall our discussions of space and bandwidth in section 3.1 on p. 24). With appropriate use of Web page links and some software assistance, you can make this kind of structure almost transparent, while spreading the load across multiple systems and networks and maintaining the integrity of any security zones you have in place.

14. John loves RRDtool. Dan agrees that it is great for data for which you need only limited historical records.

3.3.1 Everything Gets Old

Whenever you're collecting data (or log files), you really should think ahead. Everything gets old, and when it gets old it either gets useless, starts filling up your disks, or both. So when you're planning how to store your monitoring data, you should also take the time to plan how you will deal with the data as it accumulates and gets old. You wouldn't want to have to depend on a "disk full" warning from your monitoring data to remind you to tidy up after yourself, would you?

There are several ways to deal with your older data:

> **File Compression**: Most log files compress very well, and, depending how you store it, your monitoring data may also compress very well. It's likely worthwhile to compress your older files (after a month? a year?). We typically use `gzip --best`, trading a little extra CPU time for longterm space savings.[15] Depending on how you access your data, you might consider compressing your data more often and reading the data from the compressed file when you need it.
>
> **Data Aggregation**: You can prune your older data by aggregating multiple data points into a single representative number, much the same way that RRDtool uses consolidation functions. If you're using a database, this is likely a fairly simple operation (select, add, insert the summary, and delete the detail), but it can be a little more convoluted if you're using a text or other simple file to store your data. Three points: choose the appropriate consolidation/aggregation function, check your math, and make sure your tools will behave as expected when using the aggregated data.
>
> **Alternative Storage:** If you're storing your monitoring data on your production systems, perhaps on a redundant, high-performance SAN or NAS system, you may want to move your older data to alternative (i.e., cheaper) storage. You can move it to a cheaper, slower, non-redundant disk or copy it to a write-once, read-never DVD to file on a shelf. The point being: don't waste money saving your monitoring data beyond its "best before" date.

You can, of course, just throw away your old data, which is, conveniently, what the next section is all about. In any case, make sure you plan in advance how you're going to deal with your aging data in a way that makes sense for your organization and your needs.

3.4 Out with the Old

Sooner or later, you're going to have to face facts and throw out the trash. Is detailed information on network traffic levels five years ago really that relevant any longer? We claim there are two reasons for keeping older data:

- As a baseline for past behavior as compared to current behavior, and
- To help investigate any still-unanswered questions about the network or devices.

15. Dan feels that in general, using `bzip2 -9` will result in better compression than `gzip --best`, but that it is usually not worth the extra CPU time required. Using bzip2 without extraordinary compression arguments is often worthwhile, though.

We recommend that if space is at a premium, simply throw away data beyond a certain age (a year? two?) and get on with things. But if you have the luxury of space and a good system of organizing your data, keep everything until the volume of data becomes unmanageable, and then select your deletion policy.

4. Visualization

Being data-driven is the opposite of being opinion-based, and prescatenting your data in a visual way is the best way to make your point. Managers respect data more than opinion. Data speaks for itself, and when you let data speak for you, things go better.

For example, if you walk into your boss's office and say, "We need $6,000 for more Internet bandwidth," they might interpret that as merely one person's opinion and tell you to go away. They might even think that you just want to download more videos when they aren't around.

If, on the other hand, you walk in and make the same request and have a visualization of Internet usage and capacity, your request might get more respect. But if you walk in with the data and don't state an opinion, the data will speak for you. If the graph shows that you are at 80% capacity and it is obvious that "full" is only two months away, then your boss will be the one who realizes that more capacity needs to be purchased. The boss certainly won't doubt this conclusion if he or she is the one who saw it in the data.

If your boss doesn't draw this conclusion from looking at the data, and if the data does not make a compelling point even if you explain it, there's a good chance that you don't have a strong reason to purchase more Internet bandwidth: If the data isn't compelling, the reason isn't compelling. See [Kaushik] for more on being data-driven.

In chapter 6 we discuss having a regular daily look-see at your graphs, checking to make sure that all is well in your world, and analyzing and understanding your data and what it means, so you'll know a problem when you see it. This chapter looks at how we can take the monitoring data we have collected and make it visible using charts and graphs.

4.1 Visualization vs. Numbers

Before we talk about how to turn our numbers into pictures, we should discuss why we would want to and what the alternatives might be. The numbers we have collected, by themselves, tell the whole story.[1] How much, how often, how different. But the numbers, by themselves, don't tell the story clearly—the information can be hidden away, requiring careful review and analysis to extract the information we need. It may sound simple-minded, but the old adage of "a picture is worth a thousand words" applies here.

Taking your data and making it visible, through graphs, charts, red or green flashing displays, or any other mechanism, makes it easier for you and others to understand what's going on and to recognize when further review or analysis is needed.

1. We're greatly simplifying here, but bear with us.

The numbers are important, too (we discuss numerical and threshold-based problem detection in section 5.1), but visualization provides a very effective method for understanding your data. Visualization gives us tools to quickly acquire a comfortable gut feel about how things are going, especially when we make use of higher-level overviews of our data.

4.2 Visual Acuity

Before we actually create any graphs (or other graphical representations) of our data, we should first consider how we can best display our data to provide us with the most useful information.

With monitoring data, we will most often be looking at time series data, so we will usually want some form of representation of time in our visualizations. Most likely, we'll follow the common approach of a two-dimensional graph, with time along the *x*-axis, increasing to the right. This is because we will usually want to have an idea of what's happening now, as compared to the past and, possibly, as compared to our predictions for the future.

But don't get stuck in the idea that there is only one way to look at things. Perhaps other representations may provide you with deeper insight or a more effective understanding of your data.[2] What if you graphed your network throughput over the past year in a circle, with 0 at the center and maximum usage at the edge of the circle, and time proceeding clockwise around the circle? Is the most recent behavior the best indication of "normal" or the best predictor of future behavior? Or do you get a more interesting understanding by twisting your data around and looking at it in non-traditional ways?[3]

4.2.1 What Goes Together?

Sometimes a single data source stands alone and tells you the complete story by itself. For example, it may be that the amount of space used on the disk volume holding users' home directories is unaffected by anything else that may be going on (a university at the start and end of an academic term will likely see a different pattern of behavior).

But it's very common for multiple data sources to each tell part of the story, and to provide a clearer picture of what's going on when examined together instead of separately. The most common example is network traffic—you'll almost always see inbound and outbound traffic graphed together. Other examples can be found almost anywhere you look. Network traffic is correlated with Web site hits, and HVAC system duty cycles are likely influenced by the weather outside.[4]

But remember these four words: correlation is not causation. Just because two (or more) data sources seem to move in relation to each other does not necessarily mean that one is causing the other. Be careful not to jump to conclusions.

2. Well beyond the scope of this booklet is the excellent series of books by Edward Tufte (http://www.edwardtufte.com/), where he skillfully presents many ways of presenting data that exceed our limited grasp of data representations. If you need a compelling graph to make a point to management (as opposed to the simple yet sufficient ones for your own use), read his books.

3. John says, Don't labor under any misconceptions about my creativity—I'm strictly a two-dimensional, time to the right, rectangular graph kind of guy.

4. But see Dan's personal experience in section 6.2 to know that it's not always strictly related.

4.2.2 One of These Things Is Not Like the Others

Just because two data sources seem to have some sort of similarity, it doesn't mean that you should necessarily graph them together—you may end up with a graph that's confusing or misleading. You probably don't want to graph data sources that are "independent" (in the mathematical sense) on the same graph. For example, a graph of the aggregate CPU utilization across a compute server cluster is probably useful, but a graph combining the aggregate CPU utilization of two unrelated clusters probably isn't.

4.2.3 What Sort of Visualization?

We suggested above that you shouldn't get stuck in one way of thinking about graphs—they don't all have to be the same old left-to-right line graphs. Consider the data sources you're interested in, how they are or are not related, and how you can best represent those relationships. For example, traffic across a set of Web servers might best be represented in a "stacked" graph, showing the usage of individual servers (or groups of servers) and also showing the aggregate total across the set. Can you use legends, data values, or threshold indicators on your graphs to provide easy-to-understand clues to the data? Can you use different colors to represent different states (e.g., draw your network usage in red when it's over a certain threshold)? Can you provide different representations of different aspects of your data sources in one graph?

SmokePing[5] is a great example of effectively using color and different representations of the data on a single graph. SmokePing is typically used to show network latency behavior by graphing the results of sets of 10 ping packets to a network location. It graphs the average round-trip time, the number of packets lost, and the distribution of round-trip times in a clear and easy-to-understand way using color and different shades of gray.

4.2.4 Time Is on Your Side

If you're visualizing time series data, you'll likely have to choose (implicitly or explicitly) a time period to represent. You may be tempted to graph the longest time period you have available, on the assumption that more information is always better. But before you do, consider what the most common use of your visualizations is likely to be. In many cases, the most common use for graphs is to get a quick sense of what's going on now,[6] and it's often going to be the case that too much information could obscure current problems (e.g., a year-long graph may not provide enough detail to see indicators of current problems).

[Burgess02] showed the existence of strong weekly patterns as the strongest and cleanest signal of measured data over any time scale. If you want to visually detect anomalies, you should arrange your data in weekly periods, as this will reveal anomalies with the highest accuracy.

4.2.5 Consistency

Don't take what we've said above as an argument against consistency: random differences in representation just for the sake of being different would likely be confusing and counter-productive. You're likely best off representing similar things in similar ways. It will

5. Also by Tobi Oetiker and Niko Tyni: http://oss.oetiker.ch/smokeping/.

6. See chapter 6 again.

make review and understanding quicker and easier. (But don't be afraid to do something different when it's appropriate.)

4.3 Creating Visuals

There are of course many different tools for converting numbers into visuals. The tool (or tools) you choose to use will likely depend on several factors:

- How is your data stored? Some tools work primarily (or most easily) on flat files, some work with data in a database, and RRDtool (of course) works with RRD files.
- What system platform are you using? Some visualization tools are cross-platform, while some work only on certain operating systems.
- When and how do you plan to generate the graphics? As needed through a Web interface, or through periodic batch processing? Some tools are particularly well suited to one method or the other.

We will introduce a few of the most popular methods for creating graphs and other visuals, and we will refer you to other sources for more detail.

One consideration with most of these methods is that you're likely going to have to select the data you want to plot outside of the tool itself, and feed it only the data you want graphed; it's more difficult to choose just the data and time-frame you want to graph from within these tools. In general, whatever tools you use to create your visuals, you will find that the design pattern is typically:

database → filter → compute → graph

The "database" can be a query or simply cat-ing a file; "filter" can be a condition on a database query, but can also be grep, awk, Perl, or Python; "compute" can be null (if it is part of the filter, or if you don't want to post-process your data), but it can also be another awk, Perl, or Python; and "graph" is one of the tools described below.

4.3.1 Visuals with Gnuplot

Gnuplot[7] has a long history (since 1986) and supports a wide variety of operating systems and output types and devices. It reads data from text files and can generate many different graph types. It is primarily command-driven, can be scripted or batched, and can be called from a CGI script to generate graphs for the Web.

```
set title 'Gnuplot Graph for Today'
set xdata time          # x values are times
set timefmt "%s"        # seconds since epoch
set format x "%R"       # HH:MM
set xtics 1800          # mark x-axis every half hour
set terminal png size 400, 200
set output 'gnuplot.png'
plot 'data1.txt' using 1:2 title 'one' with linespoints, \
  'data2.txt' using 1:2 title 'two' with linespoints
```

Figure 4.1: Simple Sample Input to Gnuplot

7. http://www.gnuplot.info/.

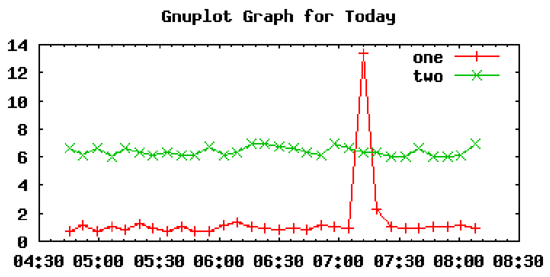


Figure 4.2: Simple Sample Output from Gnuplot

Figure 4.1 shows a very simple sample of gnuplot commands that create a PNG file (figure 4.2) from data points in two different files. You can get much more complex, of course, but with gnuplot you can get nice results without a lot of work.

4.3.2 Perl Visuals with GD::Graph

GD::Graph[8] is a graph-plotting module for Perl. It can create different types of graphs (lines, bars, etc.) but has fewer features and options than gnuplot. But if you're already Perl inclined, GD::Graph is likely a very good choice.

Figure 4.3 shows a simple example of how to use GD::Graph, with the resultant output in figure 4.4. And, like gnuplot, there's more complexity available if you want it, but you can get good results fairly easily.[9]

```
#!/usr/local/bin/perl
use GD::Graph::linespoints;
@data = read_data_from_file( “data12.txt” );
$my_graph = new GD::Graph::linespoints( 400, 200 );
$my_graph->set(
   title => ‘GD::Graph Graph for Today’,
   markers => [ 4, 4 ],        # 4 == diagonal cross
   x_tick_number => 4,
   y_tick_number => 3,
   transparent => 0,
 );
$my_graph->set_legend( ‘one’, ‘two’ );
open( IMG, ‘>gdgraph.png’ ) or die $!;
print IMG $my_graph->plot(\@data)->png;
close( IMG );
```

Figure 4.3: Simple Sample Input to GD::Graph

8. See CPAN at http://search.cpan.org/search?query=gd::graph and some additional information at http://www.gdgraph.com/.

9. Dan's thermd package uses GD::Graph—you can see more examples of GD::Graph output in chapter 6.

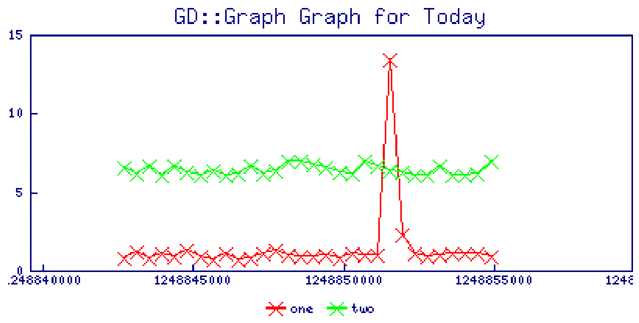


Figure 4.4: Simple Sample Output from GD::Graph

4.3.3 Visuals with PHP

There are a number of graphing packages for use with PHP and, as far as we can see, there doesn't seem to be an obvious leader in the category.

```
<?php
include( 'PHP_GNUPlot.php' );
$p = new GNUPlot();
$p->set( 'xdata time' );            // x values are times
$p->set( 'timefmt "%s"' );          // seconds since epoch
$p->set( 'format x "%R"' );         // HH:MM
$p->set( 'xtics 1800' );            // mark every half hour
$p->set( 'terminal png size 400, 200' );
$p->set( 'output "php_gnuplot.png"' );
$p->set( 'multiplot' );             // multiple data sets
$p->setTitle( "PHP_GNUPlot Graph for Today" );
$data1 = PGData::createFromFile( '/tmp/data1.txt', 'one' );
$data2 = PGData::createFromFile( '/tmp/data2.txt', 'two' );
$p->plotData( $data1, 'linespoints', '1:2' );
$p->plotData( $data2, 'linespoints', '1:2' );
$p->close();
?>
```

Figure 4.5: Simple Sample Input to PHP-GNUPlot

PHP-GNUPlot[10] is a simple PHP wrapper around the gnuplot command, but it does provide a reasonable, if incomplete, abstraction. Figure 4.5 is a simple example. The output is, not surprisingly, the same as the gnuplot output in figure 4.2.

The GDChart[11] package is an interface to the GDChart C library (section 4.3.5) and is a PHP PECL package, so installation should be easy. As of this writing there's been only one beta release, and, regrettably, there's almost no documentation. There is also the Image_Graph package,[12] which is based on some earlier work but is currently an "alpha" release from 2006. There are also a number of commercial graphing packages available.

10. http://sourceforge.net/projects/php-gnuplot/.

11. http://pecl.php.net/package/GDChart/.

12. http://pear.php.net/package/Image_Graph/.

At this point, we lean toward using gnuplot with PHP, either directly or through the PHP-GNUPlot wrapper.

4.3.4 Graphing with RRDtool

In section 3.2.2 we discussed some of the reasons for storing your data in RRD files with RRDtool. One other major advantage of using RRDtool is that it provides an easy and effective method for creating graphs from data that is in RRD files. We showed you a typical graph from RRDtool in figure 3.1. Now we're going to look at how graphs are created and some of the options you can use.

```
rrdtool \
   graph netgraph.png \
   DEF:bytes_in1=server1.rrd:traffic_in:AVERAGE \
   DEF:bytes_out1=server1.rrd:traffic_out:AVERAGE \
   AREA:bytes_in1#00ff00:"Inbound Bytes" \
   LINE1:bytes_out1#0000ff:"Outbound Bytes"
```

Figure 4.6: Simple RRDtool Traffic Graph Generation

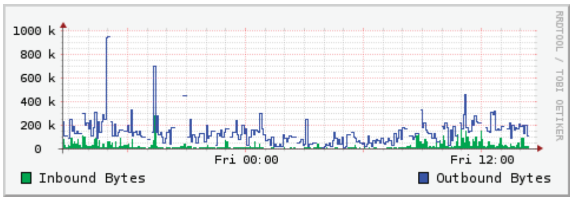


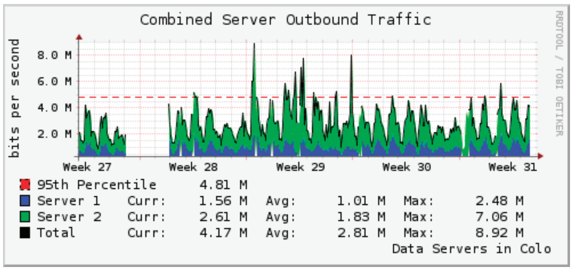
Figure 4.7: RRDtool Graph from Code in Figure 4.6

Figure 4.6 shows the rrdtool command used to create the simple graph shown in figure 4.7. RRDtool fetches two data sources (traffic_in and traffic_out) from an RRD file and assigns them to two variables (bytes_in1 and bytes_out1), and then plots them on a graph with the given style, color, and legend. You'll notice that RRDtool assumed reasonable defaults, such as the time period (the previous 24 hours) and axis labels.

Within the constraints of graphing time series data, RRDtool allows great flexibility in building and designing graphs. You can take any data sources from any RRD files, choose a time period, use "aggregation functions" (e.g., average, maximum, last) to summarize data, and use RPN arithmetic and logical expressions to calculate new data sources or values. Then you can draw them on a graph in various styles and colors and annotate the graph with almost arbitrary text and values in a variety of fonts.

```
rrdtool \
   graph netgraph.png \
   --start e-1m \
   --slope-mode \
   --title="Combined Server Outbound Traffic" \
   --vertical-label="bits per second" \
   DEF:bytes_out1=server1.rrd:traffic_out:AVERAGE \
   DEF:bytes_out2=server2.rrd:traffic_out:AVERAGE \
CDEF:bits_out1=bytes_out1,8,* \
CDEF:bits_out2=bytes_out2,8,* \
CDEF:total=bits_out1,bits_out2,+ \
VDEF:total95th=total,95,PERCENT \
LINE1:total95th#ff0000:"95th Percentile":dashes \
  GPRINT:total95th:"%7.2lf %s\l"  \
AREA:bits_out1#0000ff:"Server 1" \
  GPRINT:bits_out1:LAST:" Curr\:%8.2lf %s"  \
  GPRINT:bits_out1:AVERAGE:"Avg\: %8.2lf %s"  \
  GPRINT:bits_out1:MAX:"Max\: %8.2lf %s\l"  \
AREA:bits_out2#00ff00:"Server 2":STACK \
  GPRINT:bits_out2:LAST:" Curr\:%8.2lf %s"  \
  GPRINT:bits_out2:AVERAGE:"Avg\: %8.2lf %s"  \
  GPRINT:bits_out2:MAX:"Max\: %8.2lf %s\l"  \
LINE1:total#000000:"Total   " \
  GPRINT:total:LAST:" Curr\:%8.2lf %s"  \
  GPRINT:total:AVERAGE:"Avg\: %8.2lf %s"  \
  GPRINT:total:MAX:"Max\: %8.2lf %s\l"  \
COMMENT:"Data Servers in Colo\r"
```

Figure 4.8: More Complex RRDtool Traffic Graph Generation



```
🟥 95th Percentile      4.81 M
🟦 Server 1    Curr:    1.56 M   Avg:     1.01 M   Max:     2.48 M
🟩 Server 2    Curr:    2.61 M   Avg:     1.83 M   Max:     7.06 M
■ Total       Curr:    4.17 M   Avg:     2.81 M   Max:     8.92 M
                                          Data Servers in Colo
```

Figure 4.9: RRDtool Graph from Code in Figure 4.8

Figure 4.8 shows the command to generate the more complex RRD graph shown in figure 4.9. Many of the arguments are self-explanatory, but a few notes are likely worth making:

- The start time for the graph is set to one month before the end time. The end time defaults to "now," but a different date can, of course, be specified.
- The –slope-mode option tells RRDtool to smooth out the graph, rather than using the more accurate bar graph style, as some people prefer this style.
- A CDEF is a calculated definition providing values across the selected time range, and a VDEF calculates one value (the 95th percentile value in this example) that can be drawn across the graph. Calculations are defined using RPN, for stack-like calculations.

You will note that the resultant graph has a gap at the end of week 27; RRDtool has rules to determine when values are "undefined" and doesn't plot any values when a data source is undefined.

If you plan ahead when you create the RRD file, RRDtool will even calculate expected values for you and let you draw graphs that highlight "unexpected" values.[13]

RRDtool provides multiple mechanisms, other than the command line, for working with RRD files:

- rrdcgi can be used to generate simple Web pages and (pre-defined) graphs on the fly.
- rrdtool will accept commands on standard input, so you can generate multiple graphs in a single invocation.
- This means that you could also set up an RRD server on a TCP port using inetd, as long as you're not interested in much in the way of security or access controls.
- There are also Perl modules, RRDs (from the RRDtool distribution) and RRD::Simple,[14] to make it easy to use RRDtool from Perl, and there is librrd.a[15] for calling from C programs.
- You can use the `rrdtool fetch` command to copy data points from an RRD file for use with other programs.

Drraw—Ad Hoc Web Interface to RRDtool

Drraw[16] is a Web based front end to RRDtool that lets you create graphs on the fly, save the graph templates for later use, and also create "dashboard" views of the saved graphs. Thus you can organize related graphs or views into a single Web page and provide an easy way to get an overview of the things you're interested in.

The complication is that, while drraw is high on functionality, it requires a good understanding of how RRDtool works and how and where the data is stored, and it takes a certain amount of effort to define non-trivial graphs. But drraw provides a convenient mechanism for designing and building graphs from RRD files; you could build everything from the command line, but the Web interface adds a layer of convenience.

If you are already collecting RRD-based data with another tool (such as Cricket, section 7.1.2), then drraw dashboards (combined with tabbed browsing in your favorite Web browser) can be indispensable for visualizing your system state with a single click. On Dan's network (with five hosts, three wireless access points, a router, and a 48-port switch), a single click opens six dashboards which show just about everything he needs to know about the state of his system. In general, each machine has its own row of graphs, and each column has similar graphs across each machine.

13. See [Brutlag00] for more information.

14. http://search.cpan.org/~nicolaw/RRD-Simple-1.44/lib/RRD/Simple.pm.

15. Although librrd.a is very lightly documented.

16. http://web.taranis.org/drraw/.

For example, one dashboard tracks free disk space (with every disk on a machine shown in different colors on one graph) and disk activity (with each disk a different color, reads graphed upwards from the centerline, and writes graphed downwards), as can be seen in figure 4.10. Since Cricket typically only draws graphs with a few closely related data items on them (such as the statistics for a single disk), drraw can be very valuable for grouping less-related data together (such as statistics for a collection of disks).

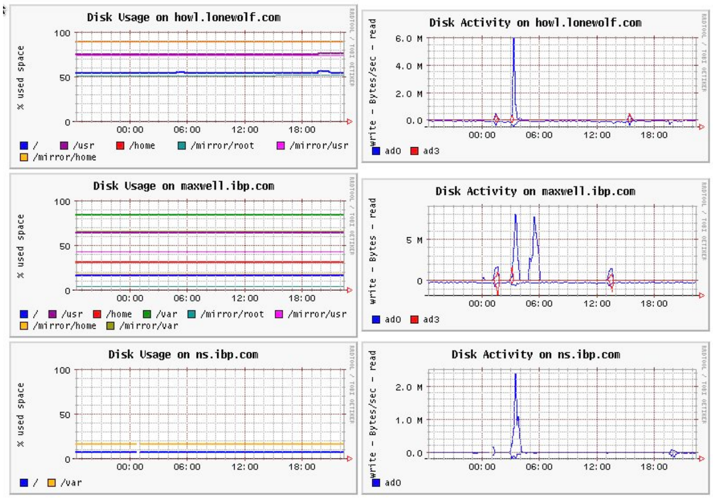


Figure 4.10: Sample Drraw Dashboard

Orca: Using RRDtool with Other Data

Orca[17] is a tool for plotting columnar, time series data from text files onto Web pages, using RRDtool to generate the graphs. Orca uses a configuration file that tells it:

- ❖ Where to find the data, and how the data is organized
- ❖ How to create the RRD files for the data
- ❖ What graphs to create, with which data

When Orca is run (periodically, if run as a daemon), it checks for new data in the data files, updates the RRD files, and generates the necessary graph images and Web pages.

If you have similar files from multiple systems (e.g., periodic statistics from the system or an application), the Orca configuration file makes it easy to specify, through a regular expression, how to find the files and system names, and Orca will generate per-system graphs and summary Web pages.

If you already have systems or applications generating time series data, Orca is a very effective way of turning those numbers into visuals.[18]

17. http://www.orcaware.com/orca/.

18. John keeps thinking that Orca is so useful that it deserves to be more widely used than it appears to be.

4.3.5 Other Visualization Tools

There are a couple of other graph generation tools that are worth a mention here.

There is, of course, a Perl CPAN module that is a front end to gnuplot: the Chart::Graph::Gnuplot[19] module, which seems to be up-to-date and full-featured.

GDChart[20] is an easy-to-use, high-performance library/C API for creating charts and graphs in gif, png, jpeg, and wbmp formats. It's currently under-documented, labeled "beta," and unchanged since 2004. We mention it here because it is the engine for GD-Chart modules for PHP, Perl, and Python, and has some utility. GDChart seems to be less featureful than gnuplot; you may be better off with gnuplot.

The RTG Real Time Grapher [Beverly02] is an SNMP statistics-monitoring package. We mention it here because it includes the rtgplot command, which generates graphs from data in a MySQL database. Its graphs are reminiscent of RRDtool-style graphs, using terse command-line arguments. Because rtgplot is already geared toward time series data, if you already have your data in a database, you may want to see if it meets your needs even if you're not interested in using RTG.

Grace[21] is a 2D plotting tool for UNIX-like systems that can be used from the command line (grace), in batch mode (gracebat), or through an X GUI (xmgrace). It will generate various output formats, including png, jpeg, and pdf (among others). Grace has a long heritage, being descended from Xmgr, which was first released in 1991. It has a reasonable feature set but may have a slightly steeper learning curve than some other options.[22]

If you want to build charts from a URL, Google Charts[23] provides the quality and versatility you'd expect from anything Google. We also recommend looking at the Google Visualization API[24] and the rather amazing Motion Chart[25] visualization, an animated Flash-based tool for time-series representation. However, for each of these tools, since all of your data points are passed to Google, you pay more in terms of network bandwidth than you do for generating a chart locally and transmitting just the image.

Timeplot[26] is a DHTML-based AJAXy widget for plotting time series and overlaying time-based events on them. Less data has to transit over the Net than with Google Charts, and Timeplot is able to add a mouseover slider that shows you the values of each line in your chart where it intersects with the mouse line (thermd also uses Timeplot).

19. http://search.cpan.org/~caidaperl/Chart-Graph-3.2/Graph/Gnuplot.pm.

20. http://www.fred.net/brv/chart/.

21. http://plasma-gate.weizmann.ac.il/Grace/.

22. We must admit that we haven't used Grace and that we find it a little strange that the documentation doesn't include any sample output.

23. http://code.google.com/apis/chart/.

24. http://code.google.com/apis/visualization/.

25. http://code.google.com/apis/visualization/documentation/gallery/motionchart.html.

26. http://www.simile-widgets.org/timeplot/.

5. Alerting

Periodically examining the data you have recorded is invaluable for determining trends and anomalies, but there are certain kinds of anomalies that you'd like to know about sooner rather than later. A smoke alarm or heat sensor that alerts us to a nascent fire is far more valuable than the discovery of the smoldering ruins of a building the next morning.

5.1 Thresholds and Problem Detection

Although the human in the loop is far more adept at noticing unusual patterns of activity, even a third-rate monitoring system is useful for detecting values that exceed reasonable thresholds.[1] Such data readings can alert us to an assortment of conditions with varying degrees of severity. They include:

- Power outages (even if you have a UPS, since you also need to know when the batteries are sufficiently depleted that a clean shutdown is your only remaining option)
- Overtemperature (e.g., equipment that is running too hot, fire detection)
- Undertemperature (e.g., danger of pipes freezing and rupturing)
- Water level (e.g., flooding, vessel overflow)
- Network load (e.g., DDoS attack for a high load, server failure for a low load)
- Server load (e.g., spam attack on a mailserver, slashdotted pages for a Web server)
- Mail queue size (e.g., a failed MTA or a spam attack)
- Door status (e.g., unauthorized entry)
- Disk status (e.g., free space low on a critical service, activity too high on a low-usage machine)

Threshold-based exceptions typically fall into one of four basic categories:

Instantaneous: An instantaneous threshold is one that is considered at every data reading (whatever that frequency may be; see section 2.3 for additional notes on what "instantaneous" may mean).

Averaged: An averaged threshold is one in which a series of data readings, considered in aggregate, exceeds a predetermined value. Averaged thresholds[2] allow

1. Binary sensors are included in this definition—a door sensor that is normally closed may have exceeded its reasonable value when it is open.

2. Averages should generally be computed as a moving (or "running") average, not a windowed average.

for normal variations in sensor readings (whether they are physical or computed values), and do not unnecessarily raise alarms.

Duration-based: A duration-based threshold is one in which readings stay above a specified value for a predetermined period of time. Typically, readings must initially exceed a trigger value without falling below a reset value. For example, a temperature/duration threshold exception is one in which the temperature initially exceeds 90° F and then remains above 85° F for 10 minutes—if the temperature drops below 85° F before 10 minutes have elapsed, the duration sensor is reset until the trigger is again reached. The trigger and the reset value can be the same value, so a threshold exception would exist so long as the temperature exceeded 90° F for 10 minutes.

Cumulative: A cumulative threshold is one in which the count of readings above a value is significant. There are a number of variations on this type of sensor. The most common one is where more than *n* readings in a row exceed a threshold, but variations can consider more than *n* readings in a time period,[3] or a majority of readings in a time period.

Except for binary state sensors, instantaneous readings are probably not a good indicator of a problem. For example, binary states such as a door opening, a smoke alarm, or a motion detector are good thresholds to pay immediate attention to. A single excessive temperature reading may indicate an exceptional condition, but we have found that the 1-Wire sensors can occasionally give an erroneous reading (which should not be considered grounds to raise an alert). A single reading of a high network load may simply indicate a normal surge in activity, and a high mail queue size may simply indicate a user who has sent a single email with a large recipient list that has not yet cleared the queue. A better approach for anything other than a binary state is to consider thresholds for either an averaged reading or a reading that has remained high for a period of time. Whether you use a duration-based or a cumulative threshold depends on the variability of the data and the sensitivity that is desired in the exception detection logic.

Regrettably, although many measurement possibilities exist for determining when a value is over threshold, the majority of monitoring systems opt for the more simple (and potentially less discriminating) instantaneous or averaging thresholds.[4]

5.2 Alerts, Methods, and Notifications

So what can we do with exceptional conditions? Detecting them is one thing, logging them is extremely useful, but getting the human into the loop is often the desired action, so that the erroneous condition can be examined and corrected.

A number of mechanisms exist for logging the exception and notifying the people who can address the problems.

3. That is, a metabolizing value (or time-weighted average), where individual values over the threshold become less significant over time, but a cumulative count of over-threshold values is significant.

4. SmokePing provides a pattern-matching mechanism to alert on ongoing problems but ignore "one-time" errors. Nagios will repeat a failing check multiple times (if configured to do so) before deciding that it's a "real" problem.

Email alerts: Perhaps the most obvious way of getting the human into the loop is to send email. Email is the "universal" communication medium, but it is subject to users checking their mailboxes.[5] The key is to "poke" the user in an asynchronous manner, so that the body of the email (which is checked synchronously) details the symptom, but the asynchronous notification comes by some other means. However, email is not guaranteed to be an instantaneous event, but may be delayed by hours or days, depending on network and server status.

SMS alerts: There exist SMS gateways wherein a server on your network can be used to transmit an SMS message directly to a user's phone. Additionally, most cell phone providers give each phone number a separate email address,[6] and when mail is sent to that address, the user's cell phone receives an SMS message. Note that email is still the bottleneck in this scheme, so email delays may still exist, and any number of problems may block the email path from your monitoring server to the user's phone.

Telephone calls and pagers: Systems can also be built to dial a telephone and/or a pager,[7] and if you have the appropriate hardware, an audio file can be streamed to the connection. It is also possible simply to use the caller ID of the dialing phone to convey the information necessary to the user being called (and if a VoIP line is used, the caller ID can often be spoofed and set to something intended to indicate the problem).

syslog: The usual method for recording an anomalous data reading is to log it via syslog. The biggest problem is that users typically do not read their logfiles unless they know that something is amiss, so simply logging the anomaly is insufficient for the purpose of alerting the administrator. It is, however, useful so that at a later time, a record exists that not only documents the event but is "surrounded" by other events in the log that may be related.

SNMP alerts: Many hardware devices with SNMP interfaces can be configured to send an SNMP "trap" to an SNMP trap handler. Additionally, systems that monitor SNMP data from routers, switches, etc., can be programmed to likewise send traps. The action of the trap handler can be to send an email, execute a remote command, etc. (basically, any of the other actions listed in this section), but it is worth mentioning the SNMP trap handler as a way of centralizing alarm handlers in a network.

SNMP traps are sent via UDP, so there is no guarantee that the trap handler will actually receive the trap, however, especially if the trap is reporting on a network problem. You'll likely want to augment your trap handling with periodic polling.

Physical actuators (horns, lights): For staffed premises, the quickest way to notify users of an exceptional event is to sound a loud horn and blink the lights.

5. Of course, we are cognizant of the fact that some users check their email on a frighteningly regular basis from their Blackberrys, iPhones, etc.

6. For example, 4125551212@@txt.att.net.

7. In fact, if you have an old modem lying around, it can trivially be converted into a simple autodialer to send pages.

This tends to send staff members scurrying about like cockroaches in a just-lit kitchen, may engender a more severe response than is desired, and, if there are a sufficient number of false alarms, may also cause users to completely ignore the alarms. Additionally, special hardware often needs to be purchased (or built) to actuate the alarms. Use these only for the direst of alerts.

6. Analysis

Recording your systems and environmental data is merely the first step in a monitoring process. The second step is that you must examine the data on a regular basis! While there certainly exist programs for monitoring (such as Cfengine and assorted intrusion detection systems), there really is no substitute for at least one set of human eyeballs looking at your data on a regular basis.

One of the daily habits that both your authors have acquired is to quickly scan the graphs of our systems and look for "something odd." We rarely find anything, but when something doesn't jibe, we look closer for possible problems.[1]

The human visual system is masterful at detecting patterns (or, alternatively, aberrations that deviate from a regular pattern).[2] Simply by looking at your data graphs on a daily basis, you will quickly recognize what looks normal and, correspondingly, what looks "odd."

Once you see something that "doesn't look quite right," you need to look at the raw data to determine the reason. Perhaps there was a passing surge (e.g., a user sent or received an unusually large email) or perhaps it is the sign of something more onerous (e.g., one of your users is running a peer-to-peer file-sharing program). The only way you can know for sure is to look at the raw logfiles (or sometimes, in the case of environmental sensors, the physical environment being monitored). If a disk in one computer is running hot, one might infer that a case fan has failed. You won't know until you look, and the problem might be caused by something else entirely!

In this section we will show you several real-world examples of where logging, monitoring, and diligence have detected problems in our "systems" (and we use that term rather loosely), and how detecting these initially small problems can help keep them from turning into larger problems.

6.1 Web Site Security Flaw

As he summarizes below, in 2006 Dan's systems were targeted in a proxied Web-based spam attack (described in detail in [Klein06]).

Basically, there was a flaw in an old and long-forgotten "request for information" email script on one of my sites, and that flaw allowed a malicious third party to append hun-

1. John wonders if doing a daily scan is a practice that scales well as a network gets larger and more complicated. He tries to rely more on automated detection of unusual conditions, through threshold or service monitoring and automated alerts. Dan generally agrees, but counters that in complex networks and environments, data can be consolidated in dashboards or stacked in aggregated graphs with drill-down hotlinks.

2. If you don't believe us, remember that the visual cortex is a massively parallel image- and signal-processing system that has evolved over millions of years—it might still have a few bugs left (that's why optical illusions still work on us), but on the whole it is a pretty robust system.

dreds of Bcc: addresses to the mail headers. This resulted in my mail machine becoming a spammer.

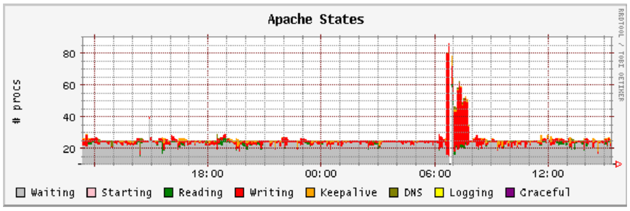


Figure 6.1: Apache States

The morning of the attack, I had received a number of odd mail messages from the script that was being attacked, but I wrote those off as "someone trying to mess with the site" (and was unconcerned, given that I thought my scripts were secure). I also received one email message for each of the actual attack messages (since the script was designed to send me that mail). However, after the first few messages (each one having 380–390 Bcc addresses), my desktop spam filters helpfully blocked those messages,[3] and except for the SNMP data, I was otherwise unaware of the thousands of spam messages being sent to AOL from my machine. Because the messages were not sent in a flood (in fact, AOL was deferring all of the messages after the first thousand or so), my network load looked fairly normal.

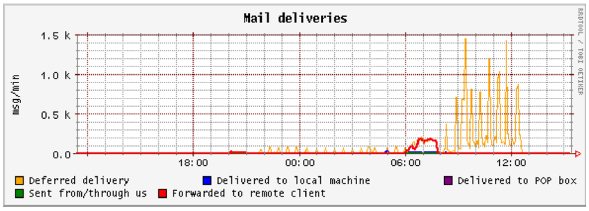


Figure 6.2: Mail Deliveries Made by Attacked Host

What ultimately triggered further analysis was the graph of Apache states, found in figure 6.1 (however, this was only after first noticing the highly unusual mail statistics in figures 6.2 and 6.3). Mine is a lightly loaded Web server, and the configuration sets MinSpareServers and MaxSpareServers to 15 and 25, respectively. Typically, fewer than a half-dozen servers are active serving requests. Thus when Apache reported 40 to 80 active processes for a span exceeding an hour, I determined that something was amiss.[4]

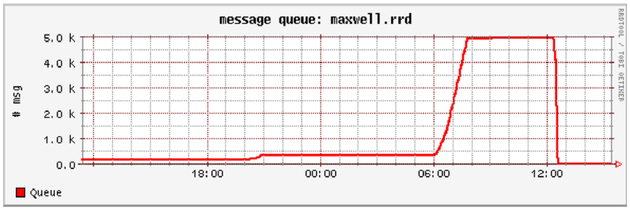


Figure 6.3: Number of Messages in the Outbound Mail Queue

3. They were blocked because the message was sufficiently spammish, and it had been seen enough times to trigger my filters.

4. The sudden drop in message queue size is when I detected and fixed the problem in the CGI script and manually cleaned the outbound mail queue.

It is especially noteworthy that there was no surge in the requests to the Web server, nor in the amount of network traffic the Web server was generating. Thus each writing process (shown in figure 6.1) was taking a lot of time to perform its job, but did so without generating a lot of data. Since Apache does not block on locked files, the only reasonable explanation was CGI script execution, where each script was taking a long time to complete. Indeed, there were a large number of active processes. The load average on the machine (which is typically below 1.0) surged to a high of 16 before Sendmail's `RefuseLA` safeties cut in, and the number of active processes nearly quadrupled, from 200 to 700.

6.2 HVAC Problems

Take a look at figure 6.4. Notice how the first-floor sensor (which is situated in the cold-air return of a gas forced-air heating system) is very hot for part of the day? It had me worried for a while, because the cold-air return should closely parallel the overall ambient temperature of the house. At first I thought that the cat who usually sleeps on the hot-air outlet vent was diverting warm air around the corner to the cold air return, but when I went downstairs, there was no cat, and warm air was gently blowing out of the cold air return (when it should have been sucking air *into* the grating). There was something clearly wrong with the HVAC system, but what?[5]

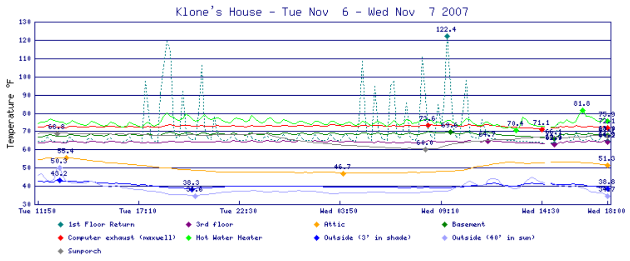


Figure 6.4: Temperature in Dan's House

I have twinned high-efficiency furnaces: a pair of 100,000 BTU furnaces tied together. One was having trouble starting, as you can see in figure 6.5.[6] Sometimes the power consumption for the furnace has a "knee" in the spike seen when the furnace is blowing hot air, and in some instances the power consumption is lower while the hot air is blowing out of the cold-air return (the anomaly in power consumption is fairly small, and I only noticed it after I found the main problem, below).

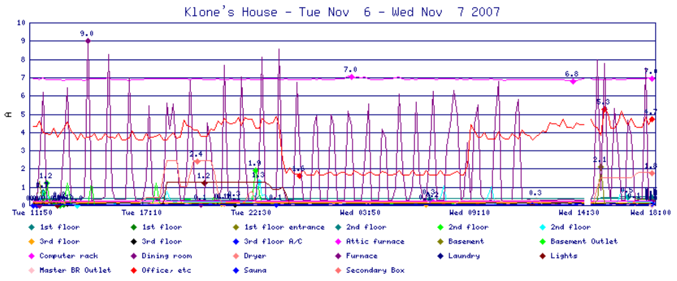


Figure 6.5: Electricity Usage in Dan's House

5. The live data for this graph is at http://www.thermd.com/thermd.cgi?from=20071106T1200&to=+30h.

6. This can be found at http://www.thermd.com/thermd.cgi?from=20071106T1200&to=+30h&view=amps.

Each furnace has its own cold-air return, but both blow hot air into the same plenum (which is distributed throughout the whole house). If the main blower fan on one of the two furnaces doesn't start, there is blowback across the plenum and up its cold-air return from the other furnace's main blower fan. And that is what I was seeing in the first graph—a blowback of hot air coming out of the cold-air return. The secondary effect of the problem was identified as an anomaly in the graphed data. So why wasn't the furnace starting?

Each of the twinned furnaces has a series of safety interlocks. For example, if you open an access panel, the main blower shuts down, so that your fingers (or your cat) don't accidentally get sucked into the whirling fan. And when the furnace ignites, a series of checks is also performed. First, the combustion fan starts. If a "prover circuit" shows that air is flowing properly, then the pilot burner circuit is activated, and a piezoelectric spark ignites a pilot light. If the pilot thermocouple shows heat is being generated from the pilot light (that is, there is a flame burning the small amount of gas needed for the intermittent pilot light), then the main burners start, and after the temperature in the combustion chamber rises high enough so that cold air will not be blown through the house, the main blower fan starts. Any single failure prevents the next system from starting (so that, for example, the absence of a pilot flame does not allow flammable gas to be released unignited into the combustion chamber). Since the main blower was not starting, something was amiss.

A manual inspection of the ignition process revealed that the combustion blower was turning but the pilot light was not igniting. Either the piezoelectric igniter had failed or the "prover circuit" was not detecting airflow.

When my gardener was here that Tuesday afternoon (6 November 2007), he cleaned up the autumn leaves with his leaf blower, and some leaf fragments were blown into the air inlet pipe for the furnace. When the combustion fan started, it sucked in the leaves, which blocked the airflow, which in turn prevented the prover circuit from registering sufficient airflow. So about 50% of the time one of the furnaces didn't start (since the leaf fragments moved around each time the blower stopped and started) or else it started late—and that accounted for the blowback.

Without monitoring, I would not have noticed this until the dead of winter, when the whole house was not heating properly. I would have wasted money on a half-functioning furnace starting and stopping, and I would have caused the working furnace to do double duty. With a regular monitoring process, I noticed and fixed the problem in less than 24 hours.

It is very important to note that we said "monitoring process" here. This includes not only the automatic data collection but also the human in the loop to look at the data! If you don't look at the data you collect (or have alerts set up), what good does it do you?

6.3 DNS Server Failover

Most network administrators, when they configure a secondary DNS server, anticipate that the secondary server will take over when the primary fails. And, of course, it is easy to test that the secondary server works by querying it directly (and equally easy to "pull the plug" on the primary server to see that your queries are answered by the secondary). But how many machines are configured to use the primary and secondary nameservers? Has each machine (either through local config files or DHCP configuration) been instructed to look at either server?

When you monitor not only machine up/down state, but also the volume of queries that are processed by each nameserver, you can authoritatively answer that question.

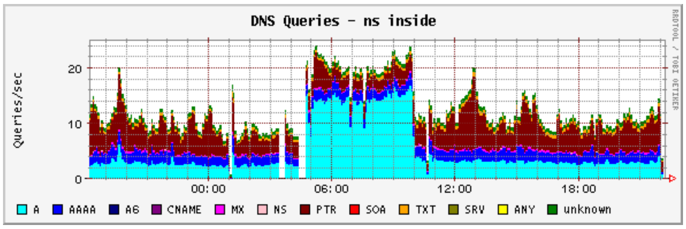


Figure 6.6: Volume of Queries on Primary Nameserver

Note the following four graphs. Figure 6.6 is the volume of queries on the primary nameserver, and you should notice three things. The first is the two dropouts between 03:00 and 05:00. This corresponds to the two peaks seen in figure 6.7, the volume of queries in the backup nameserver. Note that the heights of the peaks are roughly equal—indicating that the volume of queries being handled by the primary and secondary nameservers is approximately equal, which suggests that the two nameservers are being used equally.[7]

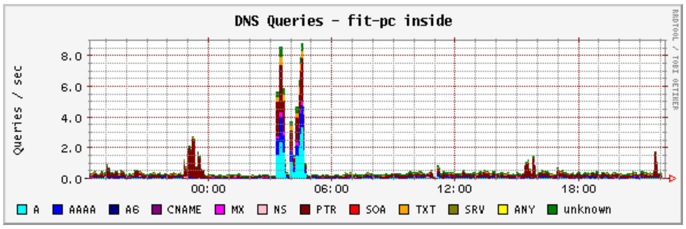


Figure 6.7: Volume of Queries on Secondary Nameserver

The second thing to notice is that the secondary nameserver handles queries for longer than the amount of time that the primary nameserver is offline. This is reasonable, as each client maintains its own cache of server availability, and once it determines that a nameserver is offline it will not continually try to contact it. But notice also that the secondary nameserver is handling queries irrespective of the status of the primary nameserver (that is, it seems to be serving PTR requests whether or not the primary nameserver is online). This might indicate a client misconfiguration or simply normal

7. Without examining every machine that uses each nameserver, we can only infer that they are all correctly configured: the stochastic nature of DNS queries could easily mask a misconfiguration of a few machines. However, from the data trends, it seems reasonable to believe that all machines are configured to use both nameservers.

It is necessary to be careful to account for the difference in scale in the two graphs, and also to note that the big surge in nameserver queries beginning at 05:45 is presaged in a ramp-up in queries in the secondary nameserver, just before the primary nameserver comes back online.

behavior. It would be necessary to examine the detailed nameserver log files to determine if a subset of client machines is making those queries, or if this is simply normal round-robin behavior.

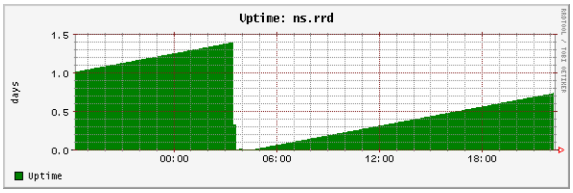


Figure 6.8: Primary Nameserver Uptime

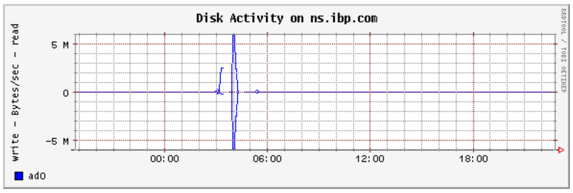


Figure 6.9: Primary Nameserver Disk Usage

Finally, the third set of things to notice is that the two peaks in the secondary nameserver queries correspond to two successive reboots of the primary nameserver. The two reboots might not be of such great significance but for the fact that, as figure 6.8 shows, the previous uptime of the primary nameserver is a mere 1.5 days! This could indicate a larger problem that needs to be addressed.[8] Figure 6.9 shows the disk utilization on the primary nameserver, and the nameserver reboots seen in figure 6.6 correspond to the file system dumps that are being run every night. Which, of course, reveals a greater problem: if dumps cause system crashes, how do we back up our system?

The answer is that the dumps succeed about 50% of the time, and until we can buy a better power supply (or change from an internal disk to solid-state storage), we're safe enough. Our nameserver configuration doesn't change that often, and the main things we are backing up are the local logfiles. We are aware of the problem, know its cause (and its solution), and are willing to live with the compromise. Hardly an ideal situation, but with monitoring at least we know more than we would if we were not monitoring!

8. In fact, in this example the primary nameserver is a Soekris box with an internal hard drive—and the power supply for the unit is insufficient for the task of concentrated disk activity, which often causes crashes when a file system dump is performed.

7. Software and Hardware Sources

Once you have decided that you want to be monitoring your systems' and environment's status, where do you get the hardware and software needed to accomplish this? We'll discuss a few of the major systems and suppliers and then encourage you to do a bit more research on the Web to find the solution that is ideal for your setup.

7.1 Software

There are many software packages that are intended to be used for the collection and visualization of time series data. We're going to mention a few of the most significant here. These packages all have their strengths and weaknesses, but they have a number of features in common:

- They have their roots in data collection via SNMP, but they can all be used to collect arbitrary data, through file access or running external commands.
- They tend to be biased toward data collection from devices on a network and to organize the information and visuals they provide on a per-device basis.
- Visuals are most often presented in a view including daily, weekly, monthly, and yearly graphs (which clearly reflects the influence MRTG has had in this area).

7.1.1 MRTG—The Multi Router Traffic Grapher

MRTG[1] [Oetiker98], as its name implies, was originally a tool for graphing the traffic passing through router or switch ports. MRTG reads a configuration file, polls the appropriate devices for the appropriate SNMP OIDs, saves the values in a file, and generates graphs and Web pages for display.

In a reflection of its history, MRTG plots two values on each graph, most commonly the inbound and outbound traffic levels on a network port. But MRTG can plot anything, and can collect data using methods other than SNMP as well. For each interface or pair of data sources, MRTG generates daily, weekly, monthly, and yearly graphs, an idea that has since been copied by many other tools.

MRTG has been enhanced in many ways over the years (notably, to start using RRDtool for data storage and graph generation) but has remained consistent with its initial, straightforward idea. As a result, MRTG is very simple to implement and needs very

1. MRTG is another tool by Tobi Oetiker, and as far as we know, was the first of its kind, dating back to spring 1995. See http://oss.oetiker.ch/mrtg/.

little maintenance once installed. One downside of MRTG is that it doesn't scale as well as some of the more recent tools, but it still excels at its original purpose.

7.1.2 Cricket

Cricket[2] [Allen99] was released in 1998 and was intended to be a high-performance replacement for MRTG that used RRDtool as its storage and visualization engine. Cricket draws on the strengths of MRTG and adds many new features, including a hierarchical configuration structure, simple mechanisms for dealing with large numbers of similar devices, options to customize just about everything, and the ability to plot any number of data sources on a single graph.

The result is a system that is much more flexible and scalable than MRTG, both in configurability and at runtime. Cricket is configured via text files (which is often a good thing) and provides a number of mechanisms for managing configurations and for automatically generating configuration fragments.

Cricket has been unchanged in recent years but is still a good choice, especially for larger environments that can benefit from automated configuration.

7.1.3 Cacti

Cacti[3] aims to be the complete RRDtool-based graphing solution. The key differentiating feature of Cacti is that it is configured and managed through a comprehensive Web interface. Some extensions require behind-the-scenes file management, but most operations are performed through the Web interface.

The major benefit of the Web interface is greater ease of use for more users, and the ability to easily control access to subsets of the configuration and data for different users. The obvious disadvantage of the Web interface focus of Cacti is that it is more difficult to automate a Cacti installation or take advantage of auto-discovery mechanisms or an existing device database.

Cacti is also not as easy to extend as other tools, as it requires additional information and detail in order to integrate with the Web interface. Adding a new type of data source can require more effort in Cacti than in other tools.

Cacti is a good choice for moderately sized networks where standard functionality would provide most (or all) of the desired results.

7.1.4 Torrus: Data Series Processing

Torrus[4] is a framework and set of tools for high-performance collection and management of time series data. It is geared more toward the functionality and scalability required in large networks, and it is expected that most configurations will be automatically generated, through inventory management tools or through device discovery mechanisms.

2. http://cricket.sourceforge.net/.

3. http://www.cacti.net/.

4. http://www.torrus.org/.

The (apparent) snag with Torrus is that the price of efficiency and scalability is a more complicated and less obvious installation, configuration, and maintenance cycle. At present, Torrus is likely best suited for larger networks, especially those with a high degree of homogeneity, such as ISPs or large organizations with a central networking group.

7.1.5 Thermd

Thermd[5] is an environmental monitoring system that supports dozens of different manufacturers' hardware. Originally designed to monitor server temperature, it now interfaces with a very large spectrum of environmental sensors, as well as speaking 1-Wire SNMP, MODBUS, serial, and Ethernet protocols. Since one of the authors of this booklet (Dan) wrote thermd, it is naturally one of our favorites—however, the "links" page on the thermd Web site lists a wide variety of other software.

7.1.6 Time Series Data Through Other Tools

In this chapter, and throughout this booklet, we have primarily assumed that the collection, analysis, and visualization of time series data will be performed through tools especially aimed toward those tasks.

There are other monitoring tools that can be used to collect time series data, but we tend to believe that single- or special-purpose tools are the best approach to dealing with time series data.

In particular, there are a number of add-ons and extensions to Nagios that are intended to extend Nagios to provide graphs of time series data and its behavior over time. The advantages of these approaches are typically claimed to be:

- Ease of configuration; you can configure service checks and time series data collection in a single place.
- Fewer checks; you can get both service state and performance data from a single test.

We tend to believe in separate tools for monitoring for exceptions and for performance —we believe that the extra configuration and management effort and the duplication of some tests are justified by the higher quality of the results. For example, time series data can best be analyzed if the data is collected at regular and predictable intervals. Nagios's service check scheduling mechanisms are geared toward providing effective service check results, and do not happen at consistent and predictable times and time intervals.

7.2 Hardware

For system and network monitoring, most modern hardware[6] has built-in mechanisms for looking at network load, port status, etc., through SNMP or similar mechanisms. However, for environmental monitoring, you'll need to purchase special hardware that typically has no other function but to monitor the environment (the exceptions to this rule are computers, routers, and switches that also provide temperature measurements for the purpose of self-health checking).

5. http://www.thermd.com/.

6. That is, the hardware you will already have in place to run your servers and network.

When deciding what hardware to buy, there are typically four axes you must consider:

Measurement-only vs. managed: Most managed environmental monitoring systems have a "pretty" human-readable Web interface (in addition to a machine-queried one), which may often include Flash or JavaScript. They also typically have built-in alarms, where they send emails, SMS messages, or SNMP traps when sensors exceed user-specified limits.

Measurement-only environmental monitoring systems typically provide only a machine-queried interface, although some provide a prettier human-readable one, too. Any alarms must be managed by external software.

Sensor types: Most environmental monitoring systems support a single sensor architecture. The most prevalent is the set of Dallas Semiconductor 1-Wire sensors, which cover a wide spectrum of measurement types (temperature, humidity, switch, light, voltage, etc.). Some use custom-built circuitry to measure other things (voltage, power factor, phase, etc.).

If you buy a managed monitoring system, you will be constrained to the devices that they support, so it is best to plan ahead and purchase the system that supports what you anticipate you will need.[7] Don't be afraid of a heterogeneous monitoring solution, where you use different vendors' hardware to measure different things.

Number of sensors: In general, managed monitoring systems typically have a (small) fixed number of sensors they can handle. Even when using 1-Wire bus technology that can support dozens of sensors per bus, they use a single sensor per bus to ensure reliable configuration and to minimize user-error issues when constructing a sensor network.

In general, measurement-only monitoring systems are more flexible in terms of the number of sensors they support (that is, when using a bussed sensor technology). Because the configuration of the sensors is in external software, the hardware can afford to be more general in nature.

Ease of use: Managed environmental monitoring systems are often easier to use (they are designed to be as close to plug-and-play as possible), but sacrifice flexibility and typically have a fixed (and small) number and type of sensors available.

Measurement-only environmental monitoring systems are usually harder to set up and configure (they all require external software to address hardware configuration and to read sensor data), but generally have more flexibility in the type and number of sensors supported.

7. For example, just about every environmental monitoring system can measure temperature, but very few can monitor wind speed and direction (which requires not only special sensing hardware—an anemometer and wind vane—but the data collector to read them). If you don't care about the wind, your choice of hardware is greatly increased. If you do care about wind measurements, you might want to simply purchase one specialized piece of measurement-only hardware to monitor it, and use managed monitoring systems to monitor the temperature of your systems and environment.

7.2.1 Hardware Vendors and Recommendations

Naturally, there is no one best environmental monitoring system to recommend—your decision is based on how important all four of the above considerations are to you. However, we can recommend a number of reliable, rugged, and powerful systems. For more information on other systems, look at the "comparisons" and "links" pages on the thermd Web site.

Embedded Data Systems HA7Net: The HA7Net from Embedded Data Systems[8] is a measurement-only, Ethernet-based, 1-Wire data collector with three separate busses. The firmware is field-upgradable, and the device can communicate with all the 1-Wire sensors currently made (some devices are read/written using built-in firmware, but all devices can be accessed using direct firmware access to the 1-Wire protocol). The data collector features device discovery and a debug interface in addition to the usual Web-based data collection mechanisms.

Dan uses the HA7Net to measure temperature, humidity, barometric pressure, wind speed and direction, door status, rainfall, sunlight, and lightning measure (and many other devices are supported). Although they only sell a limited number of these sensors (see HobbyBoards for another alternative), the Embedded Data Systems products are solidly and ruggedly built and are highly reliable.

HWgroup Poseidon: The Poseidon products from HWgroup[9] are managed, Ethernet-based 1-Wire data collectors. Like most managed systems, they only support temperature and humidity devices using 1-Wire, but they also support RS-485 sensors (which enables you to create a managed bus of up to 31 sensors). Like many other collectors, they provide dry-contact switch readings, but unlike many others, they also provide asynchronous notification of changes—an essential component for status-change alerts or security applications.

Dan uses the Poseidon 3268 to monitor temperature, humidity, and door status, and to send alarms when doors open or when the temperature exceeds a reasonable value.

OWFS and DS9097: If you have a Linux machine (or some BSD variants) and you want to perform environmental sensing on the cheap, then OWFS combined with any of the many USB-based DS9097 1-Wire interfaces is the way to go. You will need your own monitoring software, as the DS9097 is purely measurement-only (OWFS is supported by thermd).

HobbyBoards: If you are looking for inexpensive or unusual sensors, HobbyBoards[10] is a good supplier. They will sell you kits or completed sensors to measure a wide variety of environmental conditions.

TrendPoint EnerSure: If you want to measure power consumption, most measurement devices will measure a single circuit (or appliance) or will monitor the

8. http://www.embeddeddatasystems.com/.

9. http://www.hw-group.com/index_en.html.

10. http://www.hobby-boards.com/.

whole premises as an aggregate amount. However, the EnerSure from TrendPoint[11] will measure between 22 and 88 individual branch circuits in a panel. It has an unmanaged MODBUS interface, but TrendPoint has Windows-based interface software (and the device is also supported by thermd). Wiring the sensors to the device is non-trivial (it requires a licensed electrician), but it is a powerful asset for an enterprise.

Dan uses the EnerSure to monitor his main circuit breaker panel and plans on installing a second one for his secondary box.

Quality Kits QK145: If you only want to monitor temperature, have almost no budget (and are probably borrowing this booklet from a friend), then the QK145 from Quality Kits,[12] the cheapest sensor system around, is probably the one for you. It can report on up to four temperature sensors through an RS-232 serial interface. It is a kit (only) and comes with a single sensor (so you'll have to buy three more), but for $29.95 it is hard to beat.

7.3 External Monitoring of Utilities

There is a growing trend among utility companies in which, instead of sending a meter reader to your premises, they read your meters remotely (typically using some form of wireless technology). Many companies provide this data to the consumers through a Web interface, so it is possible for you to see how much electricity, gas, and/or water you are consuming on a daily basis. While this is certainly a poor substitute for monitoring consumption yourself (with a much finer on-site resolution), it is a means for efficiently and inexpensively looking at your total utility usage.

However, ever making things better, Google has partnered with a number of utility companies to provide near-real-time data with Google PowerMeter,[13] which will enable consumers (presumably both commercial and residential) to monitor their power consumption and adjust it to save money (and power).

11. http://www.trendpoint.com/.

12. http://www.qkits.com/.

13. http://www.google.org/powermeter/.

8. Advice

In this chapter, we'll offer you some words of advice.[1] We are both teachers, and one thing that all teachers learn early in their careers is that a good class (and a satisfying education process) follows this maxim:

- ❖ Tell 'em what you're gonna tell 'em.
- ❖ Tell 'em.
- ❖ Tell 'em what you told 'em.

In more scholarly terms, provide an introduction, body, and conclusion. And so, being at the conclusion of our booklet, we'd like to offer you some closing advice and re-stress just why it is you have been reading our words for all these pages.

8.1 General Advice on Monitoring

Monitoring is one of those tasks that can grow almost without bound. Once you start collecting information on the details of your systems and environment, you'll often find more and more details you will want to collect.

Technical people often share some common personality traits. We often want to make sure everything is perfect, and sometimes we don't want to start anything unless we know the complete solution.

In monitoring, the best approach is often to start small and expand over time, as your needs increase and as you learn more about your working environment, its quirks and problem areas. Start by identifying and tracking the most likely problem areas and work forward, incrementally, from there.

8.2 Advice on Environmental Monitoring

Monitoring your networks is "easy"—they are already wired up, and use network protocols for network monitoring. Monitoring your environment can be "hard" because you have to purchase (or construct) your sensors and wire them up to your new data collectors. Here is some advice from Dan based on hard-won experience:

> **Buy in bulk**: I bought 1000 feet of Cat 5 cable for my home monitoring project, and it is almost gone. When I bought it, I figured I'd be set for five years or so. I used it up in less than one year. If you are going to embark on a project that updates your old home, plan on using more than you ever imagined, because you'll keep getting new ideas. If you don't plan ahead, the checkout

1. There are bits of advice throughout this booklet, of course, but here we're going to be even more obvious about it.

counter at your local hardware megaplex will be your second home.[2] After you've bought as many temperature sensors as you need, you'll discover you need more. Spares are good, but they rarely replace damaged items—instead they simply fill in gaps you didn't notice before.

Get the right tools: You can use a pair of pliers as a punch-down tool, but a punch tool is so much easier! If your cable isn't working, a cable tester is a lot cheaper than remaking the cables over and over again.

Get good tools: A cheap punch tool is fine, but a quality cable crimper is better than a cheap one. No matter what tool you buy, be prepared to make mistakes, I made dozens of small Ethernet cables before I discovered that there are actually two standards for wiring them (T568A and T568B), and it would have been a good idea to pick *one* of them. I also learned that there is no universally accepted industry standard for wiring the 1-Wire sensors that I use (and that many manufacturer have chosen their own variants)!

Neatness counts: Because you will be wiring temperature and humidity (and other) sensors into odd places, it is tempting to just let the cables dangle. *Don't do that!* Label everything (because the next administrator may wonder why there is a Cat 5 cable that is not plugged into a router or switch, and just disconnect it), and use the same logic and strategy on sensor cables as you use for network cables.[3] Label *both* ends of a cable, and use twist-ties or velcro loops to secure any extra cable to the rack.

Demand often exceeds supply, so plan ahead: Even in my home office, I failed to completely anticipate the volume of future demand. I installed four Ethernet jacks (I only needed two at the time I built my office); all four are in use now. I put in four phone lines, all in use now. I had a 24-port Ethernet switch but needed to upgrade to a 48-port, and now I have only a dozen free ports left. I needed four lightning arrestors, so I bought six. (While we're on the subject of lightning: you need to put arrestors on *both* ends of your cable run, to protect against both lightning and ground surge, and to protect both your switch and the device connected to it).

Also, your monitoring needs will always grow. When you start monitoring the temperature of a rack, you'll discover the temperature gradient *within* the rack, and then you'll want to monitor each piece of equipment in the rack—should the router be on the top or the bottom, heatwise? The more you look at, the more you will want to look at.

8.3 Tell 'em What You Told 'em

Monitoring is more than just a good idea—for any process that you hope to manage, whether manually or automatically, it is *essential*. If you want to detect that something is wrong with your system, you must first know what is right. If you want to do more than

2. I've gone through 30 RJ-45 Cat 5 jacks, 75 or so RJ-45 cable ends, a few dozen RJ-12 cable ends, a large box of butt splices (until I discovered inline splices), and a big bag of zip-ties.

3. If you don't have a cable strategy, then Google-search for "Really Bad Wiring" to see why you need one.

guess about the current and past state of affairs of your systems, you must first monitor their behavior to establish baselines. And once you know what baseline behavior looks like, you must *continually* monitor to determine when an anomaly occurs.

Monitoring, as we have defined it in this booklet, is much more than the behavior of software, although that may be a large component of what you monitor. It can also include environmental components such as temperature, humidity, or door and smoke sensors. Monitoring can look at the status of software, but, more importantly, it can look at the throughput or load on hardware and software components. And the more you monitor, the faster you will see how components of your hardware and software systems interrelate.

Because of this interrelation (and interaction) of components, we generally advise that you should monitor as much as you can—really, as many different things as you can think of. Monitoring is so much more than "because we can." Once you start doing it, it becomes an essential component of keeping your systems healthy and functioning as smoothly as possible. To paraphrase Tom Limoncelli: If you're not monitoring, you're just running software.

The only thing you need to be careful of is that the computational cost imposed by your monitoring software does not become a significant component of your overall system load. In our experience, monitoring software is sufficiently lightweight that this is an easy problem to avoid.

8.4 In Closing

In this booklet we have tried to provide a practical introduction to collecting and analyzing data about your systems and the other devices that inhabit your working environment. We have suggested tools and techniques that can help you solve problems in your environment, and even detect problems before they happen.

We hope that you have found our efforts interesting and informative, and we hope that you have learned a few things or acquired a few new insights along the way.

Now we encourage you to get out and play with some of these tools. Collect some numbers, generate some graphs, find out something more about your environment, and gain some additional understanding in how to improve things and make your life easier.

But beware: sometimes a responsibility turns into an interest, then a hobby, then a habit, then an addiction . . . and then you write a book. Happy hacking!

Appendix: Sample Scripts

This appendix includes various sample scripts, code fragments, and output files that we hope will be useful to the reader. Text-only versions can be found online at http://www.sage.org/pubs/20_numbers/.

A.1 Smcount—Count Sendmail Events from Syslog

The script smcount watches for Sendmail syslog records, to count mail events. Note that it is designed to handle the specific log messages that we get with our particular configuration of spam-diverting milters, and it needs to be tweaked when we add or change milters. Monitoring is not always pretty, and this is one of those not-so-pretty cases.[1]

```
#!/usr/bin/perl
##
# smcount - Count entries from sendmail logfile.
#
# This program will read /var/log/maillog by default, but
# you can also give it an alternate name. It will assume
# that the file is periodically compressed, and checks
# for this, too.
##

use strict;
use Getopt::Std;
use File::Tail;
use File::Copy;
use POSIX ':termios_h';

use vars qw(%localhost %msg @msg
         $now $then
         $opt_o $opt_u
         );

$| = 1;

sub usage {
   my $msg = shift;
   warn ("$msg\n\n")        if $msg;
   die <<"==END==";
Usage: $0 -o outfile [-u time] [sendmail-logfile ...]
```

1. See the discussion of mail load in section 2.2.6.

```
   -o   Output file goes here (and is updated regularly)
   -u   Update of outfile frequency (seconds, def=10)
==END==
   }

getopts('o:u:')        || usage();
usage ("-o is required") unless $opt_o;
$opt_u ||= 10;
usage ("Integer value required for -u")
   unless $opt_u =~ /^\d+$/;

@ARGV = ("/var/log/maillog")        unless @ARGV;
my $logfile = shift;
my $ref = tie *LOG, "File::Tail",
   (name => $logfile, interval => 5);

open LOCAL, "/etc/mail/local-host-names";
while (<LOCAL>) {
   chomp;
   next if /^\s*$/;
   $localhost{lc($_)}++;
   }
close LOCAL;

$then = time;

@msg = qw(SPAM HAM UNKNOWN dcc deadair deferred
      delivery_local delivery_pop delivery_relay
      delivery_remote knownspammer lost_channel
      bogofilter ruleset_CheckMessageId
      ruleset_check_mail ruleset_check_rcpt
      ruleset_check_relay spamtrap unknown_user
      slammer throttle tbd2 tbd3 tbd4 tbd5);

unless ($msg{SPAM}) {
   for (@msg) {
      $msg{$_} = 0;
      }
   }

my $pid = fork;
exit if $pid;
die "Couldn't fork - $!\n" unless defined $pid;
POSIX::setsid() or die "Couldn't start new session\n";

while (<LOG>) {
   $now = time;
   if ($now - $then > $opt_u) {
      update_outfile();
      $then = $now;
      }
```

```
if (/User [Uu]nknown$/) {
    $msg{unknown_user}++;
    $msg{SPAM}++;
    }
elsif (/rejected by DCC$/) {
    $msg{dcc}++;
    $msg{SPAM}++;
    }
elsif (
    m#to="| /usr/local/bin/dccproc -E -o /dev/null"#
    ) {
    $msg{spamtrap}++;
    $msg{SPAM}++;
    }
elsif (
    m#did not issue MAIL/EXPN/VRFY/ETRN during#
    ) {
    $msg{deadair}++;
    $msg{UNKNOWN}++;
    }
elsif (/ACCESS DENIED due to SPAMMING/) {
    # Must come before ruleset=check_relay!
    $msg{knownspammer}++;
    $msg{SPAM}++;
    }
elsif (
    /ruleset=(check_(mail|rcpt|relay)|CheckMessageId)/
    ) {
    $msg{"ruleset_$1"}++;
    $msg{SPAM}++;
    }
elsif (/X-Bogosity: Yes/) {
    $msg{bogofilter}++;
    $msg{SPAM}++;
    }
elsif (/pre-greeting traffic/) {
    $msg{slammer}++;
    $msg{SPAM}++;
    }
elsif (/relay=(\w+\.)?(lonewolf|ibp).com.*stat=Sent/) {
    $msg{delivery_local}++;
    $msg{HAM}++;
    }
elsif (/to=\w+,.*stat=Sent/) {
    $msg{delivery_pop}++;
    $msg{HAM}++;
    }
```

```
        elsif (/to=<.*?@(.*?)>.*stat=Sent/
            && $localhost{lc($1)}) {
            $msg{delivery_relay}++;
            $msg{HAM}++;
            }
        elsif (/stat=Sent/) {
            $msg{delivery_remote}++;
            $msg{HAM}++;
            }
        elsif (/stat=Deferred:/) {
            $msg{deferred}++;
            $msg{UNKNOWN}++;
            }
        elsif (/lost input channel from.*to MTA after rcpt/) {
            # These usually follow User Unknown...
            $msg{lost_channel}++;
            $msg{UNKNOWN}++;
            }
        elsif (/Possible SMTP RCPT flood, throttling/) {
            $msg{throttle}++;
            $msg{UNKNOWN}++;
            }
        }

    sub update_outfile {
        if (-e $opt_o) {
            copy ($opt_o, "$opt_o.old")
                or die "Can't copy to $opt_o.old - $!\n";
            }
        open OUT, "> $opt_o.new"
            or die "Can't create $opt_o.new - $!\n";
        for (@msg) {
            print OUT "$_\t$msg{$_}\n";
            }
        close OUT;
        rename ("$opt_o.new", $opt_o)
            or die "Can't rename to $opt_o - $!\n";
        }
```

A.2 Mailstats—Sample Smcount Output

A sample of /var/log/mailstats generated by smcount:

```
SPAM                    2321928
HAM                     122702
UNKNOWN                 364195
dcc                     940490
deadair                 58682
```

```
deferred                    119419
delivery_local              18160
delivery_pop                9664
delivery_relay              84994
delivery_remote             9884
knownspammer                35
lost_channel                49574
bogofilter                  0
ruleset_CheckMessageId      0
ruleset_check_mail          66228
ruleset_check_rcpt          4104
ruleset_check_relay         358
spamtrap                    17676
unknown_user                1291883
slammer                     1154
throttle                    136520
tbd2                        0
tbd3                        0
tbd4                        0
tbd5                        0
```

A.3 Smcricket—Sendmail Stats Into Cricket

Smcricket is used to interface the smcount-generated Sendmail stats (as written to the file /var/log/mailstats) with Cricket.

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

BEGIN {
   $gInstallRoot = (($0 =~ m:^(.*/):)[0] || “/”) . “..”;
}

use lib “$gInstallRoot/lib”;
use strict;

# Test for the number of arguments
if (@ARGV) {
   print STDERR “usage: $0\n”;
   exit 1;
}

# Get the mail data
system “cut -f2 /var/log/mailstats”;
```

A.4 Querycount—Log BIND 9 Statistics

The script querycount is used to watch for BIND 9 log records and generate the file /var/log/named/querycount.[2]

```
#!/usr/bin/perl
##
# querycount - Count entries from named logfile.
#
# This program will read /var/log/named/queries by default,
# but you can also give it an alternate name.  It will
# assume that the file is periodically rotated, and checks
# for this.
##

use strict;
use Getopt::Std;
use File::Tail;
use File::Copy;
use POSIX ':termios_h';

use vars qw(%localhost %msg @msg
          $now $then
          $opt_o $opt_u
          );

$| = 1;

sub usage {
   my $msg = shift;
   warn ("$msg\n\n")   if $msg;
   die <<"==END==";
Usage: $0 -o outfile [-u time] [query-logfile ...]
   -o    Output file goes here (and is updated regularly)
   -u    Update of outfile frequency (seconds, def=10)
==END==
}

getopts('o:u:') || usage();
usage ("-o is required") unless $opt_o;
$opt_u ||= 10;
usage ("Integer value required for -u")
   unless $opt_u =~ /^\d+$/;

@ARGV = ("/var/log/named/queries") unless @ARGV;
my $logfile = shift;
my $ref = tie *LOG, "File::Tail",
   (name => $logfile, interval => 5);

open LOCAL, "/etc/mail/local-host-names";
```

2. See the discussion of DNS load in section 2.2.6.

```
while (<LOCAL>) {
   chomp;
   next if /^\s*$/;
   $localhost{lc($_)}++;
}
close LOCAL;

$then = time;

@msg = qw(A AAAA A6 CNAME M xNS PTR SOA TXT SRV
   ANY UNKNOWN);

# Restore saved values, we have huge numbers on reboot
# if we restore

## Restore saved log values, or initialize to zero
#if (open IN, $opt_o) {
#    my @in;
#    chomp(@in = map {split} <IN>);
#    if ($in[0] eq "A" && $in[1] > 0) {
#       %msg = @in;
#       }
#    }
unless ($msg{A}) {
   for (@msg) {
      $msg{$_} = 0;
   }
}

my $pid = fork;
exit if $pid;
die "Couldn't fork - $!\n" unless defined $pid;
POSIX::setsid() or die "Couldn't start new session\n";

while (<LOG>) {
   $now = time;
   if ($now - $then > $opt_u) {
      update_outfile();
      $then = $now;
   }
   if (/ IN A /) {
      $msg{A}++;
   } elsif (/ IN AAAA /) {
      $msg{AAAA}++;
   } elsif (/ IN A6 /) {
      $msg{A6}++;
   } elsif (/ IN CNAME /) {
      $msg{CNAME}++;
   } elsif (/ IN M x/) {
      $msg{MX}++;
```

```
        } elsif (/ IN NS /) {
            $msg{NS}++;
        } elsif (/ IN PTR /) {
            $msg{PTR}++;
        } elsif (/ IN SOA /) {
            $msg{SOA}++;
        } elsif (/ IN TXT /) {
            $msg{TXT}++;
        } elsif (/ IN SRV /) {
            $msg{SRV}++;
        } elsif (/ ANY /) {
            $msg{ANY}++;
        } else {
            $msg{UNKNOWN}++;
        }
    }

    sub update_outfile {
        if (-e $opt_o) {
            copy ($opt_o, "$opt_o.old")
                or die "Can't copy to $opt_o.old - $!\n";
        }
        open OUT, "> $opt_o.new"
            or die "Can't create $opt_o.new - $!\n";
        for (@msg) {
            print OUT "$_\t$msg{$_}\n";
        }
        close OUT;
        rename ("$opt_o.new", $opt_o)
            or die "Can't rename to $opt_o - $!\n";
    }
```

A.5 Querycountlog—Sample Querycount Output

Sample contents of /var/log/named/querycount as generated by querycount:

```
A          456690
AAAA       528130
A6         0
CNAME      0
MX         102563
NS         6063
PTR        795129
SOA        6767
TXT        59578
SRV        140
ANY        5
UNKNOWN    111055
```

A.6 Gather BIND 9 Statistics via Snmpd

Sample snmpd.conf file fragments for accessing BIND 9 stats (from /var/log/named/querycount) with snmpd:

```
extend .1.3.6.1.2021.200 dns-names \
   /usr/bin/cut -f1 /var/log/named/querycount
extend .1.3.6.1.2021.201 dns-counts \
   /usr/bin/cut -f2 /var/log/named/querycount
```

A.7 Ntpcricket—NTP Statistics Into Cricket

Ntpcricket is used to interface NTP stats with Cricket (using the existing ntpq program).[3]

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

BEGIN {
   $gInstallRoot = (($0 =~ m:^(.*/):)[0] || "./") . "..";
}

use lib "$gInstallRoot/lib";
use strict;

# Test for the number of arguments
if ($#ARGV != 1) {
   print STDERR "usage: $0 host peer\n";
   exit 1;
}

# Get the host
my ($host) = shift @ARGV;
# Get the peer
my ($peer) = shift @ARGV;
$peer = qr/$peer/i;


# Get the ntpq data
my (@return) = `/usr/bin/ntpq -p $host | tail +3`;

# Step through remaining lines
foreach (@return) {
   # Break lines into fields
   my($remote, $refid, $stratum, $t, $when, $poll,
      $reach, $delay, $offset, $jitter) = split ' ', substr($_, 1);
   # Print delay, offset, and jitter
   print "$delay\n$offset\n$jitter\n"
      if ( $remote =~ $peer );
}
```

3. See the discussion of NTP information in section 2.2.6.

Glossary

1-Wire Bus: A sensor and bus technology developed by Dallas Semiconductor that uses one wire for communicating data between, and sometimes also for providing power to, sensors and devices. There is also a ground wire, but the important data bits can be sent on a single wire.

Alert: An exception identified by a monitoring system that is to be reported. See section 5.2.

Cacti: An RRDtool-based data collection and graphing system that uses a Web-based interface for management and configuration. See section 7.1.3.

Cricket: A tool for data collection and graphing, which uses RRDtool for data storage and graph generation. See section 7.1.2.

CSV: A data file encoding format in which each data record is stored in a single line and the record's fields are organized in columns separated by commas.

Data Point: A single value from a data source, almost always representing a value or state (a sensor) at a point in time.

Data Source: A measurement or value that we are interested in, usually recorded over time as time series data. Examples include the temperature of a data center, the number of Web site hits per minute, or the number of bytes used on a file system. This term may have originated in RRDtool—that's where we appropriated it from.

Dispatch: To send an alert, by some appropriate mechanism, to a system administrator or a management console.

DOM (**D**ocument **O**bject **M**odel): A convention for representing and manipulating objects in tagged text, such as HTML or XML.

drraw: An extremely useful tool for graphing information from multiple RRD databases into a single graph, and for grouping collections of graphs into dashboards.

Exception: An unusual or unexpected condition identified by a monitoring system.

JSON (**J**ava**S**cript **O**bject **N**otation): A lightweight data-interchange format. It is easy for humans to read and write, and it is easy for machines to parse and generate (and is far less bulky than XML).

MIB (**M**anagement **I**nformation **B**ase): A means of defining and documenting a collection of related OIDs (**O**bject **Id**entifiers) in SNMP. See Simple Network Management Protocol.

Milter (portmanteau word for mail filter): An extension to mail transfer agents (MTA) like Sendmail and Postfix for the purpose of filtering out spam, viruses, etc.

Mission Critical: When your service absolutely, positively has to be there overnight.

MODBUS: A serial communications protocol published by Modicon in 1979 for use with its programmable logic controllers (PLCs). See section 2.4.4.

MRTG (**M**ulti **R**outer **T**raffic **G**rapher): A system for collecting (primarily) SNMP data and displaying it in graphs. See section 7.1.1.

Nagios: Nagios is a widely used host and service monitoring system that provides many features and extensions. Although some extensions make some time series data collection and analysis possible, it is not primarily a tool intended for the processing of time series data. See http://www.nagios.org/.

OID (**O**bject **Id**entifier): Used in an SNMP MIB. See Simple Network Management Protocol.

OWFS—1 Wire **F**ile **S**ystem: See 1-Wire Bus.

RPN (**R**everse **P**olish **N**otation, sometimes called Postfix notation): A mathematical notation in which operators follow their operands.

RRDtool (**R**ound **R**obin **D**atabase **tool**): A very popular system for logging and graphing time series data, originally developed for storing network monitoring data. Underlying RRDtool is the RRD file format, which provides a space- and time-efficient way to store numeric data over time and offers an easy way to aggregate or summarize older data. See http://oss.oetiker.ch/rrdtool/.

SAX (**S**imple **A**PI for **X**ML): A simple parser for XML; an alternative to DOM.

sensor: A numerical value that can be monitored. See section 2.2.

Service Level Agreement: A commitment by a service provider to a specific minimum level of service availability and performance.

Simple Network Management Protocol: A widely implemented mechanism for remotely monitoring and managing network-attached systems and devices. See section 2.2.4.

SLA: See Service Level Agreement.

SMART (**S**elf-**M**onitoring, **A**nalysis and **R**eporting **T**echnology): Used for disks.

SNMP: See Simple Network Management Protocol.

Torrus: The Data Series Processing Framework: A polling and graphing application that uses RRDtool and is intended to provide more flexibility and performance than other tools. See section 7.1.4.

Uninterruptible Power Supply: A battery-backed power supply that allows systems to keep running during power failures.

UPS: See Uninterruptible Power Supply.

USB (**U**niversal **S**erial **B**us): A common hardware interface and low-level communications protocol that has become nearly ubiquitous in recent years.

XML (E**x**tensible **M**arkup **L**anguage): Often used as a data encapsulation method for sensor data, especially when collected using HTTP. See http://en.wikipedia.org/wiki/XML.

XML-RPC: A remote procedure call protocol that uses XML to encode its calls and HTTP as a transport mechanism.

References

[Allen99] Jeff R. Allen, “Driving by the Rear-View Mirror: Managing a Network with Cricket,” *Proceedings of the First USENIX Conference on Network Administration*, pp. 1–10, Santa Clara, CA, April 1999. Cricket is a tool that lets users visualize a set of data sources over time and was written as a successor to MRTG. See the paper at http://www.usenix.org/events/neta99/allen.html and the Cricket Web site at http://cricket.sourceforge.net/.

[Allen04] Bruce Allen, “Monitoring Hard Disks with SMART,” *Linux Journal*, January 2004, pp. 74–77. This can be found online at http://www.linuxjournal.com/article.php?sid=6983.

[Anderson08] Paul Anderson, *LCFG: A Practical Tool for System Configuration*, Short Topics in System Administration #17, The USENIX Association, 2008.

[Beverly02] Robert Beverly, “RTG: A Scalable SNMP Statistics Architecture for Service Providers,” *Proceedings of the 16th Systems Administration Conference (LISA ’02)*, pp. 167–74, Philadelphia, PA, November 2002. See the paper at http://www.usenix.org/events/lisa02/tech/beverly.html and the RTG Web site at http://rtg.sourceforge.net/.

[Brutlag00] Jake D. Brutlag, “Aberrant Behavior Detection in Time Series for Network Service Monitoring,” *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, pp. 139–146, New Orleans, LA, December 2000. See the paper at http://www.usenix.org/events/lisa00/brutlag.html.

[Burgess98] Mark Burgess, “Computer Immunology,” *Proceedings of the 12th Systems Administration Conference (LISA ‘98),* pp. 283–97, Boston, MA, December 1998. See the paper at http://www.usenix.org/events/lisa98/burgess.html.

[Burgess02] Mark Burgess, Harek Haugerud, Sigmund Straumsnes, and Trond Reitan, “Measuring System Normality,” *ACM Transactions on Computing Systems* 20, pp 125–160, 2002. See the paper at http://research.iu.hio.no/papers/burgess-ACM-TOCS.pdf.

[Burgess07] Mark Burgess and Æleen Frisch, *A System Engineer’s Guide to Host Configuration and Maintenance Using Cfengine*, Short Topics in System Administration #16, The USENIX Association, 2007.

[Desai08] Narayan Desai and Cory Lueninghoener, *Configuration Management with Bcfg2,* Short Topics in System Administration #19, The USENIX Association, 2008.

[Kaushik] Avinash Kaushik, "Creating a Data Driven Culture," video: http://www.youtube.com/watch?v=OTu02Gab0Qw.

[Klein06] Daniel V. Klein, "A Forensic Analysis of a Distributed Two-Stage Web-Based Spam Attack," *Proceedings of the 20th Systems Administration Conference (LISA '06)*, pp. 31–40, Washington DC, December 3–8, 2006. See the paper at http://www.usenix.net/events/lisa06/tech/klein.html.

[Limoncelli07] Thomas Limoncelli, Christine Hogan, and Strata Chalup, *The Practice of System and Network Administration, 2nd Edition*, Addison-Wesley, 2007.

[Oetiker98] Tobias Oetiker, "MRTG—The Multi Router Traffic Grapher," *Proceedings of the 12th Systems Administration Conference (LISA '98)*, December 6–11, 1998. See the paper at http://www.usenix.org/event/lisa98/oetiker.html and the MRTG Web site at http://oss.oetiker.ch/mrtg/.

[Pinheiro07] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso, "Failure Trends in a Large Disk Drive Population," *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, February 13–16, 2007. See the paper at http://www.usenix.org/events/fast07/tech/pinheiro.html.

Index

About the Authors

Daniel V. Klein has been a computer geek for almost as long as he can remember (which is possibly more a function of his memory than his longevity). Dan is a frequent speaker and author, enjoys teaching (not only because people pay him to talk, but because he enjoys shedding light on strange subjects), and has been monitoring everything he possibly can for 15 years or more. He is the author of the popular thermd system, and the co-author of the jive text filter. He is a professional singer, has performed improvisational comedy for six years with Pittsburgh Theatresports, and has appeared in four movies, being cast (oddly enough) as a murderer, a sociopath, a zombie, and an ersatz swami (but really, he's a nice guy in real life).

His Erdös-Bacon Number is 6 (putting him on a par with Richard Feynman and slightly better than Stephen Hawking's 7). Dan has a Master's degree in Applied Mathematics from Carnegie-Mellon University (which is not as impressive as either Feynman's Ph.D. and Nobel Prize or Hawking's Ph.D. and Copley Medal) and is an award-winning photographer and a Certified Adjudicator of the International Championship of Collegiate A Cappella.

Dan and his two cats live in Pittsburgh, PA. You can find out more than you ever wanted to know about him at http://www.klein.com/.

John Sellens is also a long-term computer geek and remembers starting an introductory programming course long ago, using a teletype in a hotel in Florida connected (somehow) to a machine in Houston, but he got called away to go swimming and never finished the course. In the years since then, John has been a music store clerk, a public accountant, staff at the University of Waterloo and UUNET Canada, and somehow acquired a Master's degree in Computer Science from the University of Waterloo. He is the author of *System and Network Administration for Higher Reliability*, which is #7 in the Short Topics in System Administration series.

These days, John is the proprietor of SYONEX, supporting systems and networks, VoIP telephony, and all sorts of monitoring systems, tools, and toys. He has taught at conferences for many years, with a particular focus on topics related to system and network monitoring.

John is also an actor, a saxophone player, and the father of three wonderful children. He lives in Newmarket, Ontario.